

Nonlinear System Identification and Control Using Neural Networks

A thesis submitted

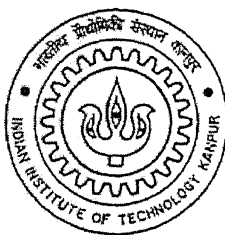
in the partial fulfillment of the requirements

for the degree of

Master of Technology

by

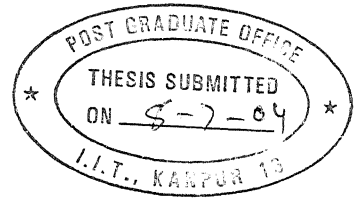
Swagat Kumar



to the

DEPARTMENT OF ELECTRICAL ENGINEERING,
INDIAN INSTITUTE OF TECHNOLOGY, KANPUR
U.P., KANPUR, INDIA.

July 2004



CERTIFICATE

This is to certify that the thesis work entitled '**NONLINEAR SYSTEM IDENTIFICATION AND CONTROL USING NEURAL NETWORKS**' by *Swagat Kumar*, Roll No. Y210440, has been carried out under my supervision for the partial fulfilment of M.Tech. Degree in the department of Electrical Engineering, IIT, Kanpur and this work has not been submitted elsewhere for any other degree.

08 July 2004

Laxmidhar Behera

Dept. of Electrical Engineering,
Indian Institute of Technology, Kanpur

TH

EE/2004/M

K96n

21 SEP 2004

मुख्योत्तम काशीनाथ केलकर पुस्तकालय
भारतीय प्रौद्योगिकी संस्थान कानपुर
अवधि क्र० A-148797.....



A148797

Acknowledgement

I would like to express my heart-felt gratitude to my thesis guide Dr. Laxmidhar Behera for providing me the necessary guidance and inspiration to carry out this work. He gave me enough freedom to experiment with my ideas and he was always there to help me out whenever I got into troubles. This work won't have been possible without his constant support and motivation.

I wish to thank Dr. K. E. Hole for all the control systems that I learnt from him during the course work. Particularly "*Nonlinear system theory*" was of great help for my thesis work.

Most of the programs of my thesis work is implemented using CVMLIB library (www.cvmlib.org), an open source common vector-matrix library written by Dr. Sergei Nikolaev. I am grateful to him for patiently answering all my queries regarding its installation and use.

A special word of thanks is due to my colleagues Bonagiri Bapiraju and Subhas Chandra Das who were always available for discussions and suggestions whenever I needed.

I also wish to express my gratitude to Awhan Patnaik for his constructive suggestions and invaluable help during my work.

Finally, I would like to extend my thanks to my friends at IIT Kanpur for their good wishes and support in various ways, especially Yashomani Kolhatkar, Milind Dighrashker, Vaibhav Srivastav and Indrani Kar.

Abstract

This thesis is concerned with system identification and control of nonlinear systems using neural networks. The work has been carried out keeping two objectives in mind. First, to design training algorithms for neural networks which are simple, efficient and capable of being implemented in the real time. Second, to design viable neural network controllers for nonlinear and underactuated systems.

Recurrent neural networks are capable of learning dynamic nonlinear systems where complete information about the states are not available. Memory Neuron Networks, a special class of RNN, has been used for identifying SISO as well as MIMO systems. The weights are adjusted using Back Propagation Through Time (BPTT). To increase the modeling accuracy, two other algorithms namely, Real Time Recurrent Learning (RTRL) and Extended Kalman Filtering (EKF) have been proposed for MNN. Simulation experiments show that RTRL provides best approximation accuracy at the cost of large training time and large training set. EKF gives comparable approximation accuracy with significant reduction in the number of presentations required as compared to RTRL.

A novel algorithm based on Lyapunov stability theory has been proposed for weight update in feedforward networks. Interestingly, the proposed algorithm has a parallel with the popular back propagation (BP) algorithm. It is shown that fixed learning rate in BP could be replaced by an adaptive learning rate which is computed using Lyapunov function approach. It is shown that a modification in the Lyapunov function can lead to smooth search in the weight space thereby speeding up the convergence. Through simulation results on various benchmark problems, it is established that the proposed algorithm out-

perform both BP and EKF algorithms in terms of convergence speed. Certain system identification issues are also analyzed for this algorithm

Some of the recent and widely known neural network based controllers have been analyzed in detail. Two existing algorithms namely NN based robust backstepping control and singular perturbation technique have been used to control various kinds of robot manipulators including flexible link and flexible joint manipulators. A neural controller based on partial feedback linearization has been proposed for pendubot. The simulation results show a promise that neural networks can be used for this class of underactuated mechanical systems which is yet to be tested through hardware implementations.

Contents

1	Introduction	1
1.1	Artificial Neural Network	1
1.2	System Identification	2
1.3	Neural Networks in Nonlinear control	3
1.4	Brief overview of the thesis	5
1.5	Memory neuron networks	6
1.6	Lyapunov based learning algorithm for feedforward networks	7
1.7	Neural controllers for robot manipulators	7
1.8	Organization of the thesis	8
I	System Identification with Neural Networks	9
2	Identification Of Nonlinear Dynamical Systems Using Recurrent Neural Networks	1
2.1	Introduction	1
2.2	Memory Neural Network	2

2.2.1	Dynamics of the network	3
2.3	Learning Algorithms	5
2.3.1	Back Propagation Through Time Algorithm	6
2.3.2	Real Time Recurrent Learning Algorithm	7
2.3.3	Extended Kalman Filter Algorithm	9
2.4	MNN For Modelling Of Dynamical Systems	10
2.4.1	Simulation	12
2.5	Summary	15
3	An Adaptive Learning Algorithm for Feedforward Networks using Lyapunov	
	Function Approach	18
3.1	Introduction	18
3.2	Lyapunov Function (LF I) Based Learning Algorithm	20
3.3	Modified Lyapunov function (LF II) Based Learning Algorithm	24
3.4	Simulation Results	27
3.4.1	XOR	28
3.4.2	3-bit Parity	30
3.4.3	4-2 Encoder	32
3.4.4	System Identification problem	32
3.4.5	2-D Gabor Function	34
3.5	Summary	35

II	Neural Controllers	37
4	Adaptive and Neural Network controllers, An analysis	38
4.1	Introduction	38
4.2	Model Reference Adaptive control of a single link manipulator	39
4.3	Self Tuning Control Based on Least Square Method	41
4.3.1	Parameter Estimation	43
4.3.2	Standard Least Square Estimator	44
4.3.3	Simulation Example	46
4.4	Neural Network based Adaptive Controller	48
4.4.1	Problem definition and stability analysis	48
4.4.2	Simulation Results	51
4.5	Summary	53
5	Neural Network controllers for Robot Manipulators	54
5.1	Introduction	54
5.2	Robot Arm Dynamics and Tracking Error Dynamics	55
5.3	Robust Backstepping Control using Neural Networks	59
5.3.1	System Description	59
5.3.2	Traditional Backstepping Design	61
5.3.3	Robust Backstepping controller design using NN	62
5.4	Singular Perturbation Design	69
5.4.1	Introduction	69

5.4.2	Singular Perturbations for Nonlinear Systems	69
5.5	Applications	73
5.5.1	Rigid Link Electrically Driven Robot Manipulator	73
5.5.2	Control Objective and Central Ideas of NN RLED Controller De- sign	74
5.5.3	Flexible Link Manipulator	81
5.5.4	Rigid-Link Flexible-Joint Manipulator	90
5.6	Summary	99
6	The Pendubot	100
6.1	Introduction	100
6.2	NN based Partial Feedback Linearization	101
6.3	Swing Up Control	102
6.4	Balancing Control	105
6.5	Simulation	107
6.6	Summary	108
7	Conclusion	109
7.1	Contributions	109
7.2	Scope of Future work	110
	Appendix	112
A	Definitions and theorems	112

A.1	Barbalat's lemma	112
A.2	Strictly Positive Real Systems	113
A.3	Zero Dynamics	113
A.4	Persistent Excitation	114

List of Figures

2.1	Structure of Memory Neuron Model	2
2.2	System identification Model	11
2.3	Plant and network output with BPTT algorithm	14
2.4	Plant and network output with RTRL algorithm	14
2.5	Plant and network output with EKF algorithm	15
2.6	Simulation results for example 2	16
3.1	A Feed-forward Neural Network	20
3.2	Adaptive Learning rates for LF-I: The four curves correspond to four patterns of the XOR problem. The number of epochs of training data required can be obtained by dividing the number of iterations by 4.	23
3.3	Adaptive Learning rates for LF-II. The four curves correspond to four patterns of the XOR problem. The number of epochs of training data required can be obtained by dividing the number of iterations by 4.	27
3.4	Convergence time comparison for XOR among BE, EKF and LF-II	29

3.5	Comparison of convergence time in terms of iterations between LF I and LF II	31
3.6	System identification - the trained network is tested on test-data	33
3.7	A 2-D Gabor function	35
4.1	Adaptive control of Single Link Manipulator	42
4.2	A self tuning controller	43
4.3	Self Tuning controller with Least Square Estimator	47
4.4	Neural Network based Adaptive Controller for single link Manipulator . .	52
5.1	Filtered error approximation-based controller	58
5.2	Two layer Neural Network	63
5.3	Backstepping NN control of nonlinear systems in "strict-feedback" form .	65
5.4	NN controller structure for RLED	77
5.5	PD control of 2-link RLED	79
5.6	NN Back Stepping control of 2-link RLED	80
5.7	Open-loop response of flexible arm	86
5.8	Neural Net controller for Flexible-Link robot arm	90
5.9	Closed loop response of flexible link manipulator with NN based SPD control	91
5.10	Closed loop response of flexible link manipulator for sinusoidal trajectory	92
5.11	Open-loop response of fast variable $q - q_m$	94

5.12	Simulation Results of NN based SPD control of 2-Link Flexible-Joint	
	Manipulator	98
6.1	The pendubot system	101
6.2	NN Controller for Pendubot	104
6.3	NN based Partial Feedback Linearization control of Pendubot	106

List of Tables

2.1	Mean Square Error while identifying with MNN	14
3.1	comparison among three algorithms for XOR problem	28
3.2	comparison among three algorithms for 3-bit Parity problem	30
3.3	comparison among three algorithms for 4-2 Encoder problem	32
3.4	Performance results for Gabor function	34

Chapter 1

Introduction

1.1 Artificial Neural Network

Artificial neural networks (ANNs) are computational paradigms which implement simplified models of their biological counterparts, biological neural networks. Biological neural networks are the local assemblages of neurons and their dendritic connections that form the (human) brain. Accordingly, ANNs are characterized by

- Local processing in artificial neurons (or processing elements, PEs),
- Massively parallel processing, implemented by rich connection pattern between PEs,
- The ability to acquire knowledge via learning from experience,
- Knowledge storage in distributed memory, the synaptic PE connections.

The attempt of implementing neural networks for brain-like computations like patterns recognition, decisions making, motor control and many others is made possible by the advent of large scale computers in the late 1950's. Indeed, ANNs can be viewed as a major new approach to computational methodology since the introduction of digital computers.

Although the initial intent of ANNs was to explore and reproduce human information processing tasks such as speech, vision, and knowledge processing, ANNs also demonstrated their superior capability for classification [1], pattern recognition and function approximation problems. Neural networks have recently emerged as a successful tool for identification and control of dynamical systems [2, 3]. This is due to the computational efficiency of the back propagation algorithm [4, 5] and the versatility of three layer feedforward network in approximating static nonlinear functions.

1.2 System Identification

System identification can be described as building good models of unknown systems from measured data. Identification of a system has two distinct steps: (i) choosing a proper model and (ii) adjusting the parameters of a model so as to minimize a certain fit criterion. Since dynamical systems are described by differential or difference equations in contrast to static systems that are described by algebraic equations, the choice of proper neural network is crucial. If all the states of the systems are available for measurement, then a multilayer perceptron (MLP) is sufficient to model the system. But in many cases, where all the states are not available for measurement or complete knowledge of dynamics

are not known, feed forward networks can not be used. Recurrent neural networks (RNN) with internal memories are capable of identifying such systems. Thus the choice of proper network model is very crucial in system identification problems. As for the second step of identification, error back propagation algorithm based on gradient descent is very popular for both feed forward as well as recurrent networks [6, 5]. Slower convergence of BP has encouraged researchers to come up with various faster convergence algorithms [7, 8, 9]. Some of the popular algorithms in this category are Lavenberg-Marquardt [10, 11], Conjugate gradient [12], Extended Kalman Filtering [13, 14]. These algorithms, although fast, are computationally intensive and require huge amount of memory. Finding weight update algorithms, which are simple and efficient in terms of computation and memory, is still a promising research area.

1.3 Neural Networks in Nonlinear control

The class of nonlinear systems that has been dealt with in this thesis include robot manipulators [15]. Robot manipulators are characterized by complex nonlinear dynamical structures with inherent unmodelled dynamics and unstructured uncertainties. These features make the designing of controllers for manipulators a difficult task in the framework of classical adaptive or non-adaptive control. Simulations and experimental results of a number of researchers such as Narendra and Parthasarathy [2, 16], Chen and Khalil [3] and many others in early 90's confirmed the applicability of neural networks in the area of dynamic modeling and control of nonlinear systems. Some other works in this field include the feedback linearization using NN by Narendra and Levin [17], dynamic neural

network for input-output linearization by Delgado et al [18], network inversion control design by Behera et al. [19]. What these works signify is that there is a growing need of algorithms that can be implemented in the real time.

Robust and adaptive controls have been used extensively to control robot manipulators [20, 21, 22]. These techniques make certain simplifying assumptions which reduce their applicability. Moreover computational requirement goes up even with moderately complex systems with many uncertain parameters. Recent literatures in robust and adaptive control report a lot of applications of some new schemes like back-stepping [21, 23] and singular perturbation [24]. The application of neural networks has been able to remove some of the limitations of classical adaptive techniques, thus broadening the class of systems that can be solved by these NN based controllers. Kwan et al. [25] have demonstrated neural network based robust backstepping control for various kinds of nonlinear systems. Similarly, Lewis et al. [26] have demonstrated various Neural Network based controllers for robot manipulators. Their work have shown that simple 2-layer feedforward networks can be used for taking into account inherent nonlinearities and parameter variations in the system thus avoiding cumbersome computations required for determining regression matrices. Moreover, these controllers can be used online owing to the simplicity of their architecture and learning methodologies.

Underactuated mechanical systems (UAMS) are a different class of nonlinear systems where the number of actuators is always less than the number of degrees of freedom. The possibility of reducing the number of actuators to control a system has attracted a lot of attention towards UAMS. Some of the well known underactuated systems are inertia

wheel pendulum, pendubot, acrobot and furuta pendulum. Robot manipulators with flexible joint and flexible links are also regarded as underactuated because of the reduced control effectiveness. Fantoni et al. [27] have proposed various energy and passivity based controllers for UAMS. Spong [28] have demonstrated a method based on partial feedback linearization for acrobot while Block [29] has demonstrated a similar method for pendubot. Quite recently, Saber [30] has suggested various methods for underactuated mechanical systems. Infact, he, for the first time, provides a classification of the underactuated systems and suggested different control schemes for different classes. Use of neural networks and other machine learning approaches for these kind of problems have not been reported so far in the literature. It seems to be an open field for research.

1.4 Brief overview of the thesis

This thesis consists of two parts. The first part deals with system identification with Neural Networks while the later part deals with design of neural network based controllers for nonlinear systems like robot manipulators.

Three different learning algorithms namely BPTT, RTRL and EKF have been used for training a recurrent network model (MNN) to identify both SISO as well as MIMO systems and a performance comparison has been made. EKF has been used for training MNN for the first time.

A new algorithm based on Lyapunov stability theory has been proposed for training feedforward networks and a performance comparison has been made with two existing popular algorithms.

A detailed analysis has been done on the recent neural network based adaptive and robust control techniques for various robot manipulators. This will give direction to my future research endeavours. Some existing algorithms like neural network based robust back stepping and singular perturbation techniques have been used to control various robot manipulators.

A neural network based controller has been designed for pendubot which is yet to be tested on a hardware set up.

1.5 Memory neuron networks

As mentioned earlier, recurrent Networks can be used for identifying dynamic networks, but deriving algorithms for weight update is quite complex because of the presence of feedbacks. Sastry et al [31] showed that by adding a temporal element to each neuron, one can achieve the functional capability of a recurrent network while preserving the simplicity of feedforward networks. They called it *Memory Neuron Network* (MNN). They proposed an algorithm based on Back-Propagation for updating weights. Some extension to their work has been done. RTRL [6] and EKF [13] have been proposed for this architecture of RNN. The efficiencies and usefulness of these three algorithms have been compared. These algorithms have been tested on both single-output and double-output systems and their performance has been assessed through various simulations.

1.6 Lyapunov based learning algorithm for feedforward networks

The training time and number of training examples required by a particular algorithm has always been a matter of concern. A training algorithm that requires minimum training exemplars and minimum computational time is always desirable. An effort has been made to come up with an algorithm which outweighs some of the best known algorithms for feedforward networks. Use of lyapunov functions for finding out suitable control input for any system is quite popular and has been demonstrated by Behera et al [32, 19]. Extending the same concept to neural networks, an weight update algorithm (LF-I) based on Lyapunov stability theory [33] has been derived. The adaptive learning rates of LF algorithm have been analyzed and this gives us the clue to avoid local minima during training. A modification in LF-I has also been suggested which is found to speed up the error convergence. A performance comparison has been made with two existing popular algorithms namely BP and EKF on three benchmark problems. System identification issues have also been discussed for the proposed algorithm.

1.7 Neural controllers for robot manipulators

Some of the recent NN based control techniques for robot manipulators have been implemented. These include robust back stepping control [25] and singular perturbation technique [26] for both flexible link as well as flexible joint manipulators. A neural network controller based on partial feedback linearization has been designed for pendubot.

Through extensive simulations, the performance of these controllers have been analyzed.

1.8 Organization of the thesis

Chapter 2 is concerned with Memory Neuron Networks and its training.

In Chapter 3, a new training algorithm based on Lyapunov function has been proposed and analyzed.

Chapter 4 concerns with classical and neural network based adaptive controls.

In chapter 5, two existing techniques namely back-stepping and singular perturbation have been used to control various robot manipulators.

In chapter 6, a Neural Network based controller based on partial feedback linearization is suggested for pendubot.

Chapter 7 gives concluding remarks and future direction of research.

Part I

System Identification with Neural Networks

Chapter 2

Identification Of Nonlinear Dynamical Systems Using Recurrent Neural Networks

2.1 Introduction

A recurrent network model with internal memory is best suited for identification of systems for which incomplete or no knowledge about its dynamics exists. In this sense, Memory Neuron Networks (MNN) [31] offer truly dynamic models for identification of nonlinear dynamic systems. The special feature of these networks is that they have internal trainable memory and can hence, directly model dynamical systems without having to be explicitly fed with past inputs and outputs. Thus, they can identify systems whose order is unknown or systems with unknown delay. Here each unit of neuron has, associated with it, a memory neuron whose single scalar output summarizes the history of past activations of that unit. The weights of connection into memory neuron involve feedback loops, the overall network is now a recurrent one.

The primary aim of this chapter is to analyse the different learning algorithms on the

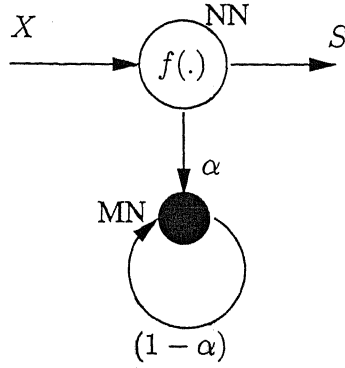


Figure 2.1: Structure of Memory Neuron Model

basis of modeling accuracy and computational complexity. The weights of MNN [31] are adjusted using a BPTT update algorithm. To increase the modeling accuracy two other algorithms namely RTRL [6] and EKF [13] have been proposed. It is concluded that RTRL identifies the system more efficiently when modeling accuracy as well as computational complexity are taken into account.

2.2 Memory Neural Network

In this section, the structure of the network is described. The network used is similar to the one that is described in [31]. The architecture of a Memory Neuron Model is shown in figure 2.1. At each level of the network, except the output level, each of the network neurons has exactly one memory neuron connected to it. The memory neuron takes its input from the corresponding network neuron and it also has a self feedback. This leads to storage of past values of the network neuron in the memory neuron. All the network neurons and the memory neurons send their output to the network neurons of next level. In the output layer, each network neuron can have a cascade of memory neurons and each

of them send their output to that network neuron in the output layer.

2.2.1 Dynamics of the network

The following notations are used to describe the functioning of the network.

L is the number of layers of the network with layer 1 as the input layer and layer L as the output layer.

N_l is the number of network neurons in layer l .

$x_j^l(k)$ is the net input to the j th network neuron of layer l at time k .

$s_j^l(k)$ is the output of the j th network neuron of layer l at time k .

$v_j^l(k)$ is the output of the memory neuron of the j th network neuron of layer l at time k , $l < L$.

$w_{ij}^l(k)$ is the connecting weight from the i th network neuron of layer l to the j th network neuron of layer $l + 1$ at time k .

$f_{ij}^l(k)$ is the connecting weight from the memory neuron of the i th network neuron of layer l to the j th network neuron of layer $l + 1$ at time k .

$\alpha_j^l(k)$ is the connecting weight from the j th network neuron to its corresponding memory neuron.

$\alpha_{ij}^L(k)$ is the connecting weight from the $(j - 1)$ th memory neuron to the j th memory neuron of the i th network neuron in the output layer at time k .

$v_{ij}^L(k)$ is the output of the j th memory neuron of the i th network neuron in the output layer at time k .

$\beta_{ij}^L(k)$ is the connecting weight from the j th memory neuron of the i th network neuron in the output layer at time k .

M_j is the number of memory neurons associated with the j th network neuron of the output layer.

$g(\cdot)$ is the activation function of the network neurons.

The net input to the j th network neurons of layer l , $1 \leq l < L$, at time k is given by

$$x_j^l(k+1) = \sum_{i=0}^{N_{l-1}} w_{ij}^{l-1}(k) s_i^{l-1}(k) + \sum_{i=1}^{N_{l-1}} f_{ij}^{l-1}(k) v_i^{l-1}(k) \quad (2.1)$$

In the above equation, we assume that $s_0^l(k) = 1$ for all l and w_{0j}^l is the bias for the j^{th} network neuron in the layer $l+1$. The output of the network neuron is given by

$$s_j^l(k+1) = g(x_j^l(k+1)), 1 \leq l \leq L \quad (2.2)$$

The activation function used for hidden and output nodes are g_1 and g_2 respectively. They have following form

$$g_1(x) = \frac{c_1}{(1 + e^{-k_1 x})} \quad (2.3)$$

$$g_2(x) = c_2 \frac{(1 - e^{-k_2 x})}{(1 + e^{-k_2 x})} \quad (2.4)$$

Here c_1, c_2, k_1 and k_2 are the parameters of the activation function. The net input to the j^{th} network neuron in the output layer is given by

$$x_j^L(k+1) = \sum_{i=0}^{N_L-1} w_{ij}^{L-1}(k) s_i^{L-1}(k) + \sum_{i=1}^{N_L-1} f_{ij}^{L-1}(k) v_i^{L-1}(k) + \sum_{i=1}^{M_j} \beta_{ji}^L(k) v_{ji}^L(k) \quad (2.5)$$

The output of all the memory neurons except for those in the output layer, are derived as follows:

$$v_j^L(k+1) = \alpha_j^L(k) s_j^L(k) + (1 - \alpha_j^L(k)) v_j^L(k) \quad (2.6)$$

For the memory neurons in the output layer,

$$v_{ij}^L(k+1) = \alpha_{ij}^L(k) v_{ij-1}^L(k) + (1 - \alpha_{ij}^L(k)) v_{ij}^L(k) \quad (2.7)$$

where by notation, we have $v_{i0} = s_i^L$. To ensure stability of the network dynamics, we impose the condition that $0 \leq \alpha_{ij}^L, \alpha_i^L, \beta_{ij}^L \leq 1$.

2.3 Learning Algorithms

Different learning algorithms to be used for the MNN are described here. At each instant, an input is supplied to the system and the output of the network is calculated using the dynamics of MNN. A teaching signal is then obtained and is used to calculate the error at the output layer and update all the weights in the network. The usual squared error is

used and is given by

$$E(k) = \sum_{j=1}^{N_L} (s_j^L(k) - y_j(k))^2 \quad (2.8)$$

where, $y_j(k)$ is the teaching signal for the j^{th} output node at time k .

2.3.1 Back Propagation Through Time Algorithm

The training algorithm using back propagation through time [5] for a recurrent net is based on the observation that the performance of such a network for a fixed number of the time steps N is identical to the results obtained from a feed forward net with $2N$ layers of adjustable weights. The final equation for updating the weights are given below:

$$w_{ij}^l(k+1) = w_{ij}^l(k) - \eta \delta_j^{l+1}(k) s_i^l(k) \quad 1 \leq l < L \quad (2.9)$$

where η is the step size and local gradient term δ_j is given by

$$\delta_j^L(k) = (s_j^L(k) - y_j(k)) g'(x_j^L(k)) \quad (2.10)$$

$$\delta_j^l(k) = g'(x_j^l(k)) \sum_{p=1}^{N_{L+1}} \delta_p^{l+1}(k) w_{jp}^l(k) \quad 1 \leq l < L \quad (2.11)$$

The above is the standard back propagation of error without considering the memory neurons. The updating of f is same as that of w except that the output of the corresponding memory neuron is used rather than the network neuron.

$$f_{ij}^l(k+1) = f_{ij}^l(k) - \eta \delta_j^{l+1}(k) v_i^l(k) \quad 1 \leq l < L \quad (2.12)$$

The various memory coefficients are updated as given below:

$$\alpha_j^l(k+1) = \alpha_j^l(k) - \eta' \frac{\partial E}{\partial v_j^l}(k) \frac{\partial v_j^l}{\partial \alpha_j^l}(k), 1 \leq l < L \quad (2.13)$$

$$\alpha_{ij}^L(k+1) = \alpha_{ij}^L(k) - \eta' \frac{\partial E}{\partial v_{ij}^L}(k) \frac{\partial v_{ij}^L}{\partial \alpha_{ij}^L}(k) \quad (2.14)$$

$$\beta_{ij}^L(k+1) = \beta_{ij}^L(k) - \eta' \delta_i^L(k) v_{ij}^L(k) \quad (2.15)$$

where

$$\frac{\partial E}{\partial v_j^l}(k) = \sum_{s=1}^{N_{l+1}} f_{js}^l(k) \delta_s^{l+1}(k) \quad (2.16)$$

$$\frac{\partial v_j^l}{\partial \alpha_j^l}(k) = s_j^l(k-1) - v_j^l(k-1) \quad (2.17)$$

$$\frac{\partial E}{\partial v_{ij}^L}(k) = \beta_{ij}^L(k) \delta_i^L(k) \quad (2.18)$$

$$\frac{\partial v_{ij}^L}{\partial \alpha_{ij}^L}(k) = v_{ij-1}^L(k-1) - v_{ij}^L(k-1) \quad (2.19)$$

Two step size parameters are used in the above equations namely, η' for memory coefficients and η for remaining weights. To ensure the stability of the network, we project the memory coefficient back to the interval (0,1), if after above updating they are outside the interval.

2.3.2 Real Time Recurrent Learning Algorithm

This algorithm [6] can be run on line, learning while sequences are being presented rather than after they are complete. It can thus deal with sequences of arbitrary length, there are

no requirements to allocate memory proportional to the maximum sequence length. The notations used in this algorithm are as follows:

$$P_{nij}^{lp}(k+1) = \frac{\partial s_p^l(k+1)}{\partial w_{ij}^n(k)} = g'_1(x_p^l(k+1)) + \sum_{r=0}^{N_{l-1}} w_{ij}^{l-1}(k) \frac{\partial s_r^{l-1}(k)}{\partial w_{ij}^n(k)} + \sum_{r=1}^{N_{l-1}} f_{ij}^{l-1}(k) \frac{\partial v_r^{l-1}(k)}{\partial w_{ij}^n(k)} \quad (2.20)$$

$$Q_{nij}^{lp}(k+1) = \frac{\partial v_p^l(k+1)}{\partial w_{ij}^n(k)} = \alpha_p^l(k) P_{nij}^{lp}(k) + (1 - \alpha_p^l(k)) Q_{nij}^{lp}(k) \quad (2.21)$$

$$R_{nij}^{pq}(k+1) = \frac{\partial v_{pq}^L(k+1)}{\partial w_{ij}^n(k)} = \alpha_{pq}^L(k) R_{nij}^{pq-1}(k) + (1 - \alpha_{pq}^L(k)) R_{nij}^{pq}(k) \quad (2.22)$$

where $1 \leq n < L$ and $1 \leq l \leq L$. It may be noted that these values at time $k = 0$ are initialised to zero. Also depending on the plant equation, these values can be re-initialised to zero after a particular number of time steps. The learning rule for this algorithm is derived as follows:

$$w_{ij}^n(k+1) = w_{ij}^n(k) - \eta \frac{\partial E}{\partial w_{ij}^n(k)} \quad (2.23)$$

$$\frac{\partial E}{\partial w_{ij}^n(k)} = \sum_{p=1}^{N_L} (s_p^L(k+1) - y_p(k+1)) P_{nij}^{Lp}(k+1) \quad (2.24)$$

where

$$P_{nij}^{Lp}(k+1) = g'(x_p^L(k+1)) \sum_{r=1}^{N_L} [w_{rp}^{L-1}(k) P_{nij}^{(L-1)r}(k) + f_{rp}^{L-1}(k) Q_{nij}^{(L-1)r}(k)] + \sum_{s=1}^{M_p} \beta_{ps}^L(k) R_{nij}^{ps}(k) \quad (2.25)$$

The update of f is same as that of w except that the output of the corresponding memory neuron is used rather than the network neuron. The various memory coefficients are updated as in previous algorithm with the difference that the learning is real time.

2.3.3 Extended Kalman Filter Algorithm

The Extended Kalman Filter uses second order training that processes and uses information about the shape of the training problem's underlying error surface. Williams et al. [34, 35] have provided a detailed analytical treatment of EKF training of the recurrent networks, and suggested a four to six fold decrease relative to RTRL in the number of presentations of the training data for some simple finite state machine problems. EKF is a method of estimating state vector. Here the weight vector $\mathbf{a}(t)$ is considered as the state vector to be estimated. The MNN can be expressed by following nonlinear system equations for i^{th} input.

$$\mathbf{a}_i(t) = \mathbf{a}_i(t-1) \quad (2.26)$$

$$y^d = h[\mathbf{a}_i(t)] + \epsilon(t) \quad (2.27)$$

Here y^d is the desired output, $\hat{y}(t)$ is the estimated output vector at time $(t-1)$ and the approximation error $\epsilon(t)$ is assumed as the white noise vector with covariance matrix $R(t)$.

The covariance matrix is unknown a priori and has to be estimated. For this purpose, $R(t)$ is assumed to be a diagonal matrix ΛI . The initial state $\mathbf{a}(0)$ is assumed to be a random vector. The following real time learning algorithm [13] is used to update the weights.

$$\hat{\mathbf{a}}_i(t) = \hat{\mathbf{a}}_i(t-1) + K_i(t)[y^d - \hat{y}(t)] \quad (2.28)$$

where K , the Kalman filter gain is given by

$$K_i(t) = \frac{1}{\lambda} P_i(t-1) H_i^T(t) \left[I - \frac{H_i(t) P_i(t-1) H_i^T(t)}{\lambda(t) + H_i(t) P_i(t-1) H_i^T(t)} \right] \quad (2.29)$$

$$\lambda(t) = \lambda(t-1) + \frac{1}{T_{max}}(t) \left[\frac{(y^d - \hat{y}(t))^T (y^d - \hat{y}(t))}{N_L} - \lambda(t-1) \right] \quad (2.30)$$

$$P_i(t) = P_i(t-1) - P_i(t-1) K_i(t) H_i(t) \quad (2.31)$$

where

$$H_i(t) = \left[\frac{\partial \hat{y}(t)}{\partial \mathbf{a}_i} \right]_{\mathbf{a}=\hat{\mathbf{a}}(t-1)} \quad (2.32)$$

Note that all the P_i coefficients for the corresponding weights are initialised to unity.

2.4 MNN For Modelling Of Dynamical Systems

A series-parallel model is obtained (for a SISO plant) by having a network with two input nodes to which we feed $u(k)$ and $y_p(k-1)$. The single output of the network will be

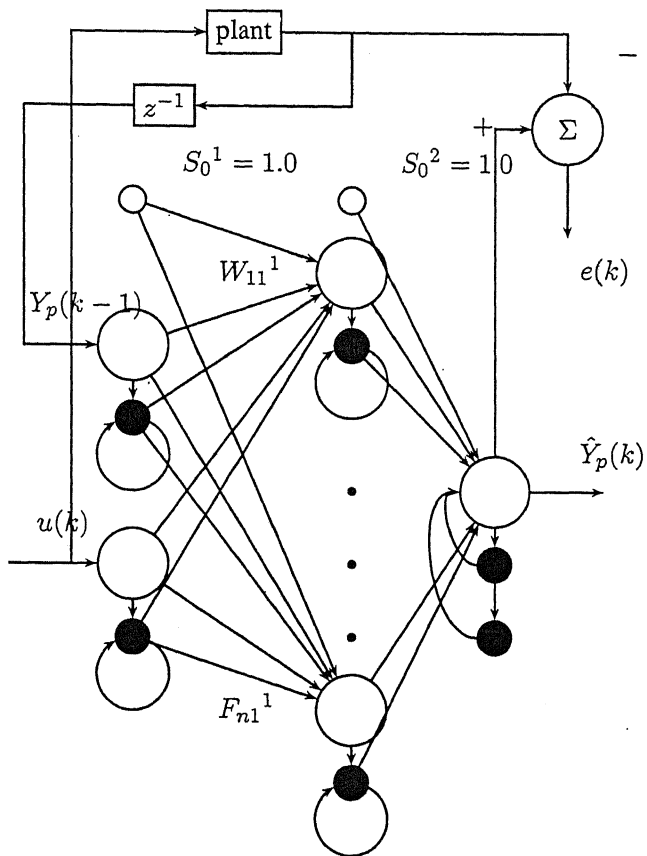


Figure 2.2: System identification Model

$\hat{y}_p(k)$. This identification system is shown in Figure 2.2.

$$\hat{y}_p(k) = F(u(k), u(k-1), y_p(k-1), \dots, \hat{y}_p(k), \dots) \quad (2.33)$$

To model an m -input and p -output plant, a network with $m + p$ inputs and p outputs will be used. This will be the case irrespective of the order of the system. The actual outputs of the plant at each instant are used as teaching signals.

2.4.1 Simulation

Two examples of nonlinear plant are identified by MNN. Series-parallel model (Figure 2.2) is used for identification. The networks with only one hidden layer are used. So, the notation $m : n$ is used to denote a network that has m hidden network neurons and n memory neurons per node in the output layer. SISO plant has two inputs $u(k)$ and $y_p(k-1)$ and output $\hat{y}_p(k)$. The number of inputs to the identification model does not depend on the order of the plant.

Network parameters: The network size used for all examples and algorithms is 6:1. The same learning rate is used for all problems with $\eta = 0.2$ and $\eta' = 0.1$. Also the same activation functions g_1 for hidden nodes and g_2 for output nodes are used with $c_1 = c_2 = k_1 = k_2 = 1$. Attenuation constant is used in the plant output so that the teaching signal for the network is always in $[-1, 1]$.

Training the network: 77000 time steps are used for training the network. The network is trained for 2000 iterations on zero input; then for two-third of remaining training time, the input is independent and identically distributed (iid) sequence uniform over $[-$

2,2] and for rest of the training time, the input is a single sinusoid given by $\sin(\pi \frac{k}{45})$.

After the training, the output of the network is compared with that of the plant on a test signal for 1000 timesteps. For the test phase, the following input is used.

$$\begin{aligned}
 u(k) &= \sin(\pi \frac{k}{25}), \quad k < 250 \\
 &= 1.0, \quad 250 \leq k < 500 \\
 &= -1.0, \quad 500 \leq k < 750 \\
 &= 0.3\sin(\pi \frac{k}{25}) + 0.1\sin(\pi \frac{k}{32}) + \\
 &\quad 0.6\sin(\pi \frac{k}{10}), \quad 750 \leq k < 1000
 \end{aligned} \tag{2.34}$$

Example 1: This indicates the ability of the MNN to learn a plant of unknown order.

Here the output of the plant is given as below:

$$y_p(k+1) = f(y_p(k), y_p(k-1), y_p(k-2), u(k), u(k-1)) \tag{2.35}$$

where

$$f(x_1, x_2, x_3, x_4, x_5) = \frac{x_1 x_2 x_3 x_5 (x_3 - 1) + x_4}{1 + x_3^2 + x_2^2} \tag{2.36}$$

Example 2: This is a MIMO plant with two inputs and two outputs. The plant is specified by:

$$y_{p1}(k+1) = 0.5 \left[\frac{y_{p1}(k)}{1 + y_{p2}^2(k)} + u_1(k) \right] \tag{2.37}$$

$$y_{p2}(k+1) = 0.5 \left[\frac{y_{p1}(k) y_{p2}(k)}{1 + y_{p2}^2(k)} + u_2(k) \right] \tag{2.38}$$

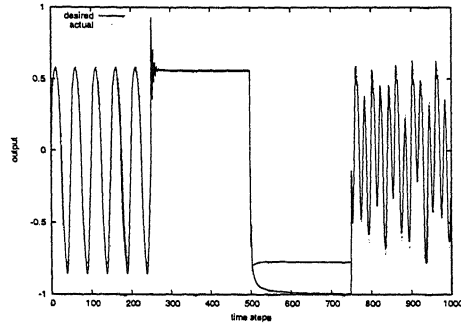


Figure 2.3: Plant and network output with BPTT algorithm

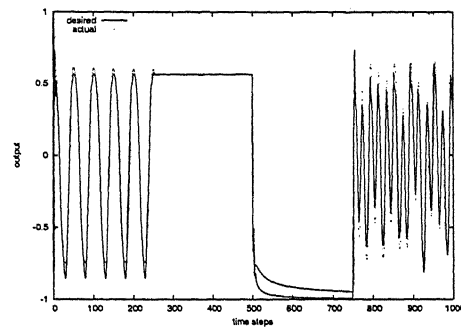


Figure 2.4: Plant and network output with RTRL algorithm

The identification is found to be good for the MIMO plant also.

Example	BPTT	RTRL	EKF
Ex.No.1	0.013293	0.006088	0.006642
Ex.No.2 o/p1	0.005753	0.001414	0.002595
Ex.No.2 o/p2	0.008460	0.001258	0.001563

Table 2.1: Mean Square Error while identifying with MNN

The examples described have been simulated using all the algorithms discussed. For example 1, figures 2.3, 2.4, 2.5 give the outputs of the plant and the model network. The plant and network outputs for example 2 is shown in the figure 2.6. The mean square

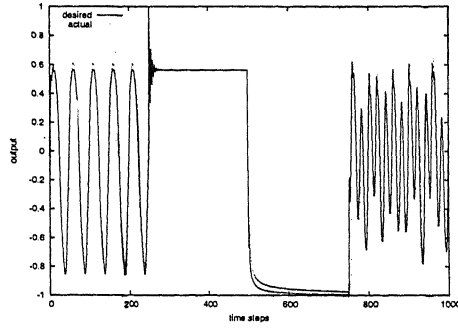


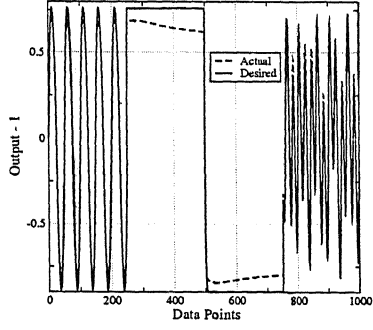
Figure 2.5: Plant and network output with EKF algorithm

errors for all the algorithms for both examples are shown in the Table 2.1.

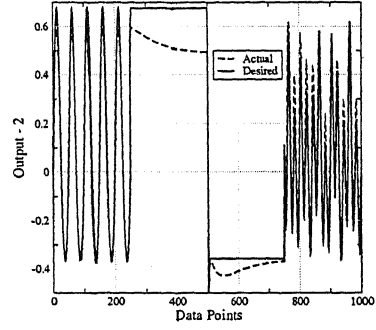
2.5 Summary

Memory Neuron Networks offer truly dynamical models. The memory coefficients are modified online during the learning process. Here the network has a near to feed-forward structure which is useful for having an incremental learning algorithm that is fairly robust. We can consider MNN to be a locally recurrent and globally feed-forward architecture that can be considered as intermediate between feed-forward and general recurrent networks.

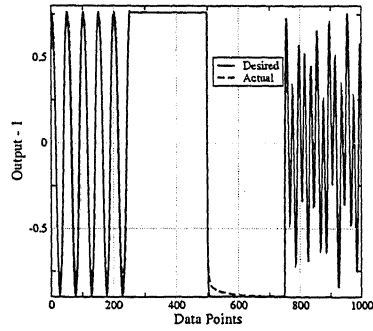
The Back Propagation through time (BPTT) algorithm is not an online training process but the real time recurrent learning algorithm is an online training algorithm with very good identification properties. The Extended Kalman Filter is a fast algorithm and shows comparable identification capabilities. It can be concluded from the graphs and error obtained in the previous section that EKF algorithm is the best suitable for modeling while the approximate gradient descent is the least favourable. The complexity of



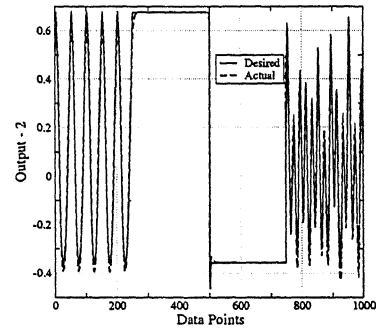
(a) BPTT: first output



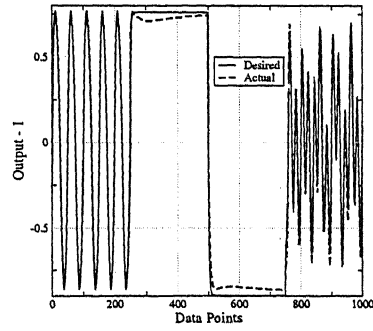
(b) BPTT: second output



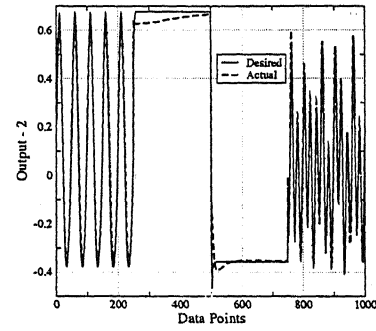
(c) RTRL: first output



(d) RTRL: second output



(e) EKF: first output



(f) EKF: second output

Figure 2.6: Simulation results for example 2

computation increases from BPTT to RTRL to EKF.

By introducing dynamics directly into the feed forward network structure, MNN represents a unique class of dynamic model for identifying any generalized plant equation. From the extensive simulations of different algorithms carried out and observing the results obtained, we conclude that EKF is one of the best learning algorithms for this model. But, from the above discussion we find that the complexity of calculations involved increases with the decrease in error. So, future research in this field will hopefully lead to improvement in that direction.

Chapter 3

An Adaptive Learning Algorithm for Feedforward Networks using Lyapunov Function Approach

3.1 Introduction

This chapter is concerned with the problem of training a multilayered feedforward neural network. Faster convergence and function approximation accuracy are two key issues in choosing a training algorithm. The popular method for training a feedforward network has been the back propagation algorithm (BP) [1, 16]. One of the shortcomings of this algorithm is the slow rate of convergence. A lot of research has been done to accelerate the convergence of the algorithm. Some of the approaches use ad-hoc methods like using momentum while others use standard numerical optimization techniques. The popular optimization techniques use quasi-Newton methods. The problem with these methods are that, their storage and memory requirements go up as the square of the size of the network. The nonlinear optimization technique such as the Newton method [12, 7] - conjugate gradient descent - have been used for training. Though the algorithm converges

in fewer iterations than the BP algorithm, it requires too much computation per pattern. Extended Kalman filtering (EKF) [13] and recursive least square (RLS) [8] based approaches which are also popular in nonlinear system identification have been proposed for training feedforward networks. Among other algorithms, Lavenberg-Marquardt (LM) algorithm is quite well-known [10, 9, 11] for training feedforward networks; although such algorithms are also computationally expensive.

In this chapter, we have proposed a very simple and easy to implement algorithm based on Lyapunov stability criterion to train a feedforward neural network. Interestingly, the proposed algorithm has exact parallel with the popular BP algorithm except that the fixed learning rate in BP algorithm is replaced by an adaptive learning rate. This algorithm was earlier used by Behera et al. [32, 19] for network inversion and controller weight adaptation as well. Yu et al. [36] have proposed a backpropagation learning framework for feedforward neural network and showed that all other algorithms like LM, GD, Gauss-Newton etc. are special cases of this generalized framework. They have also made use of lyapunov stability theory to derive the update law. They have choosen a different lyapunov function and they have not explored the nature of algorithms on different problems. Our work provides a better insight into the working of the algorithm. Three bench-mark functions, XOR, 3-bit parity and 4-2 Encoder are taken to study comparative performance of the proposed algorithm with BP and EKF. A system identification problem is also considered for testing function approximation accuracy. Lastly, we compared our algorithm with BP on a 2-D Gabor function [37] approximation problem which provides more insight into the working of this algorithm.

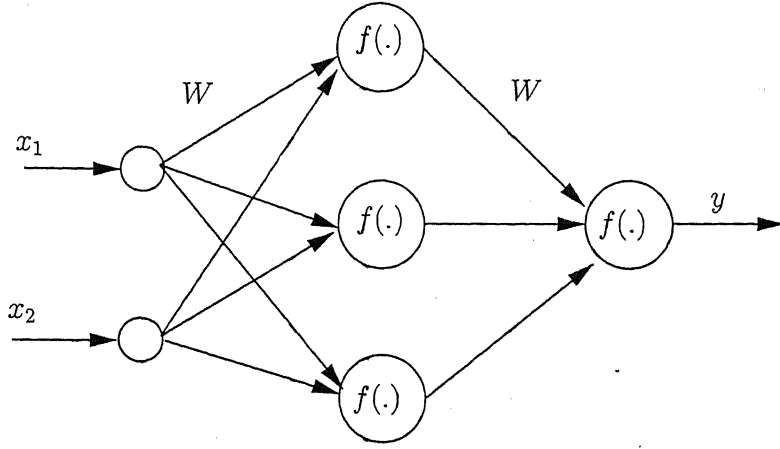


Figure 3.1: A Feed-forward Neural Network

3.2 Lyapunov Function (LF I) Based Learning Algorithm

A simple feedforward neural network with single output is shown in figure 3.1. The network is parametrized in terms of its weights which can be represented as a weight vector $W \in R^M$. For a specific function approximation problem, the training data consists of, say, N patterns, $\{x^p, y^p\}, p = 1, 2, \dots, N$. For a specific pattern p , if the input vector is x^p , then the network output is given by

$$\hat{y}^p = f(W, x^p) \quad p = 1, 2, \dots, N \quad (3.1)$$

The usual quadratic cost function which is minimized to train the weight vector W is:

$$E = \frac{1}{2} \sum_{p=1}^N (y^p - \hat{y}^p)^2 \quad (3.2)$$

In order to minimize the above cost function, we consider a Lyapunov function for the system as below:

$$V = \frac{1}{2}(\tilde{\mathbf{y}}^T \tilde{\mathbf{y}}) \quad (3.3)$$

where $\tilde{\mathbf{y}} = [y^1 - \hat{y}^1, \dots, y^p - \hat{y}^p, \dots, y^N - \hat{y}^N]^T$. As can be seen, in this case the Lyapunov function is same as the usual quadratic cost function minimized during batch update using back-propagation learning algorithm. The time derivative of the Lyapunov function V is given by

$$\dot{V} = -\tilde{\mathbf{y}} \frac{\partial \hat{\mathbf{y}}}{\partial \mathbf{W}} \dot{\mathbf{W}} = -\tilde{\mathbf{y}}^T \mathbf{J} \dot{\mathbf{W}} \quad (3.4)$$

where

$$\mathbf{J} = \frac{\partial \hat{\mathbf{y}}}{\partial \mathbf{W}} \quad \mathbf{J} \in R^{N \times M}$$

Theorem 3.1 *If an arbitrary initial weight $\mathbf{W}(0)$ is updated by*

$$\mathbf{W}(t') = \mathbf{W}(0) + \int_0^{t'} \dot{\mathbf{W}} dt \quad (3.5)$$

where

$$\dot{\mathbf{W}} = \frac{\|\tilde{\mathbf{y}}\|^2}{\|\mathbf{J}^T \tilde{\mathbf{y}}\|^2} \mathbf{J}^T \tilde{\mathbf{y}} \quad (3.6)$$

then $\tilde{\mathbf{y}}$ converges to zero under the condition that $\dot{\mathbf{W}}$ exists along the convergence trajectory.

proof: Substitution of equation (3.6) into equation (3.4), we have

$$\dot{V} = -\|\tilde{\mathbf{y}}\|^2 \leq 0 \quad (3.7)$$

where $\dot{V} < 0$ for all $\tilde{y} \neq 0$. If \dot{V} is uniformly continuous and bounded, then according to *Barbalat's lemma* [23] as $t \rightarrow \infty$, $\dot{V} \rightarrow 0$ and $\tilde{y} \rightarrow 0$. The weight update law given in equation (3.5) is a batch update law. Analogous to instantaneous gradient descent or BP algorithm, the instantaneous LFI learning algorithm can be derived as:

$$\dot{\mathbf{W}} = \frac{\|\tilde{\mathbf{y}}\|^2}{\|\mathbf{J}_i^T \tilde{\mathbf{y}}\|^2} \mathbf{J}_i^T \tilde{\mathbf{y}} \quad (3.8)$$

where $\tilde{\mathbf{y}} = \mathbf{y}^p - \hat{\mathbf{y}}^p \in R$ and $\mathbf{J}_i = \frac{\partial \hat{\mathbf{y}}^p}{\partial \mathbf{W}} \in R^{(1 \times M)}$. The difference equation representation of the weight update equation based on equation (3.8) is given by

$$\hat{\mathbf{W}}(t+1) = \hat{\mathbf{W}}(t) + \mu \dot{\mathbf{W}}(t) \quad (3.9)$$

$$= \hat{\mathbf{W}}(t) + \left(\mu \frac{\|\tilde{\mathbf{y}}\|^2}{\|\mathbf{J}_i^T \tilde{\mathbf{y}}\|^2} \right) \mathbf{J}_i^T \tilde{\mathbf{y}} \quad (3.10)$$

Here μ is a constant which is selected heuristically. We can add a very small constant ϵ to the denominator of equation (3.6) to avoid numerical instability when error $\tilde{\mathbf{y}}$ goes to zero. Now, we compare the Lyapunov function based algorithm (LFI) with the popular BP algorithm based on gradient descent principle. In gradient descent method we have,

$$\Delta \mathbf{W} = -\eta \frac{\partial E}{\partial \mathbf{W}} \quad (3.11)$$

$$= \eta \mathbf{J}_i^T \tilde{\mathbf{y}} \quad (3.12)$$

$$\hat{\mathbf{W}}(t+1) = \hat{\mathbf{W}}(t) + \eta \mathbf{J}_i^T \tilde{\mathbf{y}} \quad (3.13)$$

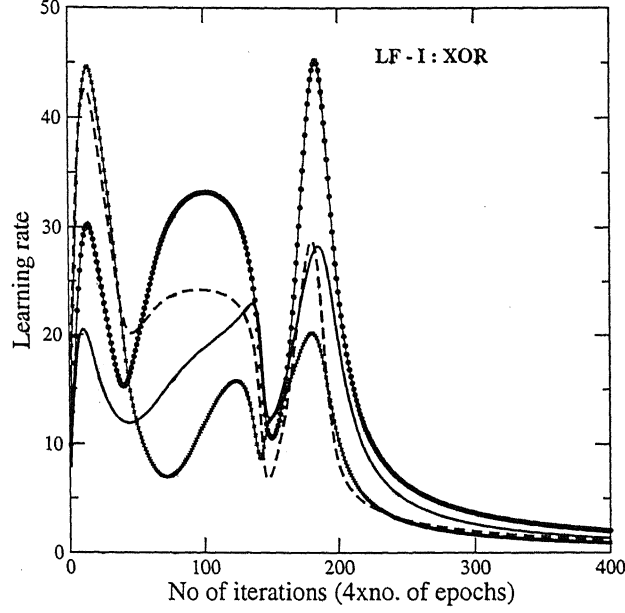


Figure 3.2: Adaptive Learning rates for LF-I: The four curves correspond to four patterns of the XOR problem. The number of epochs of training data required can be obtained by dividing the number of iterations by 4.

where η is the learning rate. Comparing equation (3.13) with equation (3.9), we see a very interesting similarity where the fixed learning rate η in BP algorithm is replaced by its adaptive version η_a :

$$\eta_a = \left(\mu \frac{\|\tilde{y}\|^2}{\|J_i^T \tilde{y}\|^2} \right) \quad (3.14)$$

Earlier, there have been many research papers concerning the adaptive learning rate [38]. However, in this chapter, we formally derive this adaptive learning rate using Lyapunov function approach, and is a natural key contribution in this field. For training XOR function, we plot adaptive learning rate in figure 3.2. It should be noted that as training con-

verges, this adaptive learning rate η_a goes to *zero* as expected. In simulation section, we will show that this adaptive learning rate makes the algorithm faster than the conventional BP.

3.3 Modified Lyapunov function (LF II) Based Learning Algorithm

In this section, we modify the Lyapunov function considered in LF I algorithm so that weight update rule can account for smooth search in the weight space. A possible Lyapunov function candidate for smooth search can be as follows:

$$V = \frac{1}{2}(\tilde{\mathbf{y}}^T \tilde{\mathbf{y}} + \lambda \tilde{\mathbf{W}}^T \tilde{\mathbf{W}}) \quad (3.15)$$

where $\tilde{\mathbf{W}} = \hat{\mathbf{W}} - \mathbf{W}$. $\hat{\mathbf{W}}$ is the ideal weight vector and \mathbf{W} is the actual weight vector. The variable $\tilde{\mathbf{y}}$ is as defined in LF I. The parameter λ is a constant. The purpose of adding a second term will be revealed later in the section. The time derivative of Lyapunov function is given by

$$\dot{V} = -\tilde{\mathbf{y}}^T \frac{\partial \hat{\mathbf{y}}}{\partial \mathbf{W}} \dot{\mathbf{W}} - \tilde{\mathbf{W}}^T \dot{\mathbf{W}} \quad (3.16)$$

$$= -\tilde{\mathbf{y}}^T (\mathbf{J} + \mathbf{D}) \dot{\mathbf{W}} \quad (3.17)$$

where $J = \frac{\partial \hat{y}}{\partial \hat{W}} : N \times M$ Jacobian matrix, $\dot{\tilde{W}} = -\dot{W}$ and

$$D = \lambda \frac{1}{\|\tilde{y}\|^2} \tilde{y} \tilde{W}^T \quad (3.18)$$

Theorem 3.2 *If an arbitrary initial weight is updated by*

$$W(t') = W(0) + \int_0^{t'} \dot{W} dt \quad (3.19)$$

where \dot{W} is given by:

$$\dot{W} = \frac{\|\tilde{y}\|^2}{\|(J + D)^T \tilde{y}\|^2} (J + D)^T \tilde{y} \quad (3.20)$$

then \tilde{y} converges to zero under the condition that \dot{W} exists along the convergence trajectory.

proof: Substituting for \dot{W} from (3.20) into (3.16), we have

$$\dot{V} = -\|\tilde{y}\|^2 \leq 0 \quad (3.21)$$

where $\dot{V} < 0$ for all $\tilde{y} \neq 0$ and $\dot{V} = 0$ iff $\tilde{y} = 0$. As derived in LF I, the instantaneous weight update equation using modified Lyapunov function can be finally expressed in difference equation model as follows:

$$\hat{W}(t+1) = \hat{W}(t) + (\mu \frac{\|\tilde{y}\|^2}{\|(J_i + D)^T \tilde{y}\|^2}) (J_i + D)^T \tilde{y} \quad (3.22)$$

To draw out a similar comparison between LF-II and back-propagation, we consider the following cost function,

$$E = \frac{1}{2}(\tilde{y}^T \tilde{y} + \lambda \tilde{W}^T \tilde{W}) \quad (3.23)$$

Using gradient-descent, We have,

$$\Delta \tilde{W} = -\eta \frac{\partial E}{\partial \tilde{W}} \quad (3.24)$$

$$= \eta (J_i + D)^T \tilde{y} \quad (3.25)$$

$$\hat{W}(t+1) = \hat{W}(t) + \eta (J_i + D)^T \tilde{y} \quad (3.26)$$

Comparing equations (3.22) and (3.26), the adaptive learning rate in this case is given by

$$\eta' = \frac{\|\tilde{y}\|^2}{\|(J_i + D)^T \tilde{y}\|^2} \quad (3.27)$$

The adaptive learning rate for LF-II in case of XOR problem is shown in figure 3.3. In equation (3.15), we define \tilde{W} to be the difference between last two consecutive weight values. By doing so, we are putting a constraint on the variation of weights. We will show in the simulation that this term has the effect of smoothing the search in the weight space thereby speeding up the convergence. This also helps in achieving uniform performance for different kind of initial conditions which is unseen in case of BP.

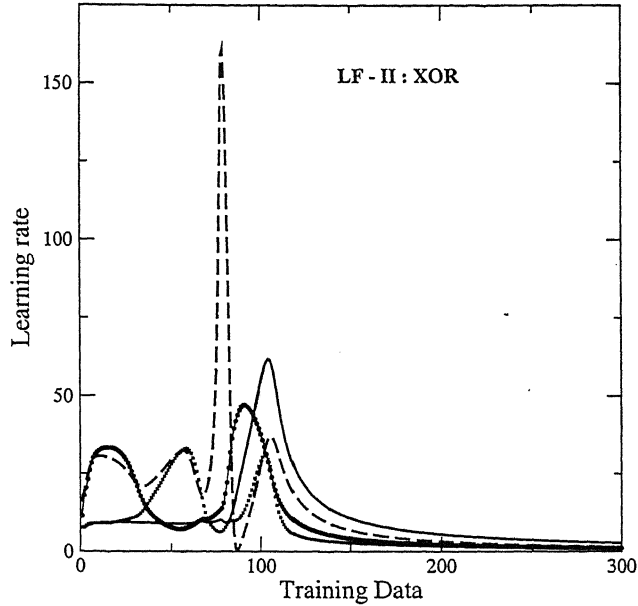


Figure 3.3: Adaptive Learning rates for LF-II. The four curves correspond to four patterns of the XOR problem. The number of epochs of training data required can be obtained by dividing the number of iterations by 4.

3.4 Simulation Results

A two-layered feedforward network is selected for each problem. Unity bias is applied to all the neurons. We test proposed algorithms LF I and LF II on three bench-mark problems, XOR, 3-bit Parity and 4-2 Encoder and a system-identification problem. The proposed algorithms are compared with popular BP and EKF algorithm. All simulations were carried out on an AMD Athlon (2000 XP) machine (1.6GHz) running on Linux (RedHat 9.0). For XOR, 3-bit Parity and 4-2 Encoder problems, we have taken unipolar sigmoid as our activation function while for system-identification problem we have

Algorithm	epochs	time(sec)	parameters
BP	5620	0.0578	$\eta = 0.5$
BP	3769	0.0354	$\eta = 0.95$
EKF	3512	0.1662	$\lambda = 0.9$
LF-I	165	0.0062	$\mu = 0.55$
LF-II	109	0.0042	$\mu = 0.55, \lambda = 0.01$

Table 3.1: comparison among three algorithms for XOR problem

chosen bipolar sigmoid activation function for neurons. The patterns are presented sequentially during training. For benchmark problems, the training is terminated when the mean square error per epoch reaches 10^{-4} . Since the weight search starts from initial small random values, and each initial weight vector selection can lead to different convergence time, average convergence time is calculated for fifty different runs. *Each run implies that the network is trained from any arbitrary random weight initialization.* In Back Propagation algorithm, the value of learning rate η is taken to be 0.95. It is to be noted that in usual cases, learning rate for BP is taken to be much smaller than this value. But in our case, the problems being simpler, we are able to increase the speed of convergence by increasing this learning rate. We have deliberately done this to show that the proposed algorithms are still faster than this. The initial value of λ in EKF is 0.9. The value of constant μ in both LF I and II is selected heuristically for best performance. Its value lies between 0.2-0.6 and λ in LF-II (3.15) is kept at a low value ($=0.01-0.1$). It simply suggests that we are giving less weightage to the second term in that equation.

3.4.1 XOR

For XOR, we have taken 4 neurons in the hidden layer and the network has two inputs.

The adaptive learning rates of LF-I and LF-II are shown in figure 3.2 and 3.3 respectively.

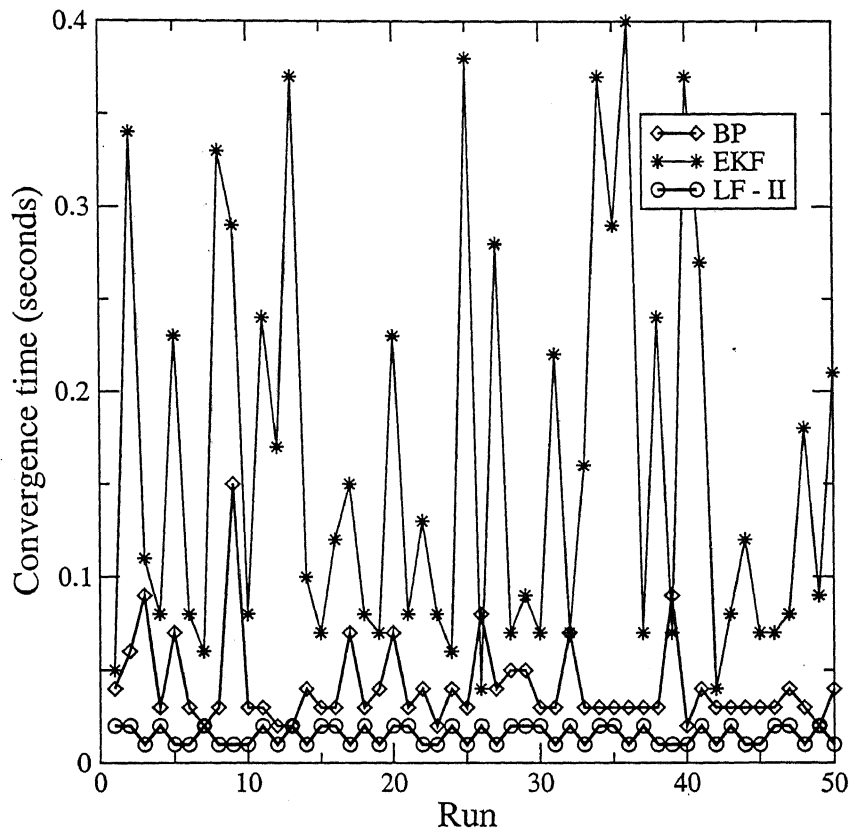


Figure 3.4: Convergence time comparison for XOR among BP, EKF and LF-II

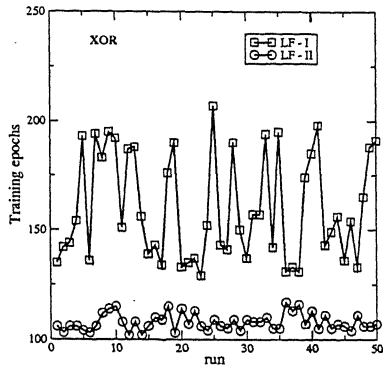
Algorithm	epochs	time(sec)	parameters
BP	12032	0.483	$\eta = 0.5$
BP	5941	0.2408	$\eta = 0.95$
EKF	2186	0.4718	$\lambda = 0.9$
LF-I	796	0.1688	$\mu = 0.53$
LF-II	403	0.0986	$\mu = 0.45, \lambda = 0.01$

Table 3.2: comparison among three algorithms for 3-bit Parity problem

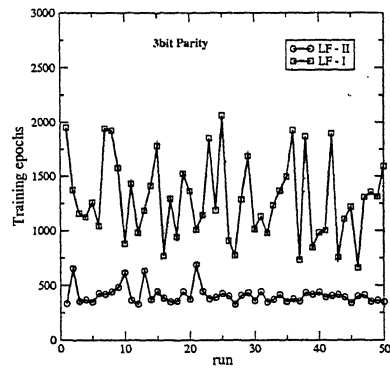
It can be seen that the adaptive learning rate becomes zero as the network gets trained. The simulation results for XOR is given the table 3.1. It can be seen that LF-I and LF-II takes minimum number of epochs for convergence as compared to BP and EKF. We find that LF-II is nearly 10 times faster than the BP algorithm. Also, we see that LF-II performs better as compare to LF-I as far as training time is concerned. The figure 3.4 gives a better insight into the performance of various networks.

3.4.2 3-bit Parity

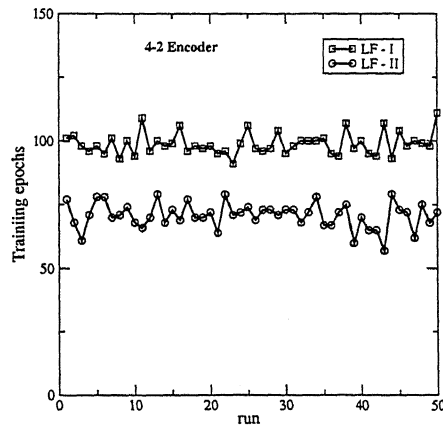
For Parity problem, we have choosen a network with 3 inputs and 7 hidden neurons. Table 3.2 shows the simulation results for this problem. Here also we find that LF-I and LF-II outperform both BP and EKF in terms of convergence time. In this case, we find that LF-II is nearly five times faster as compared to BP. EKF might be faster for a particular choice of initial condition but on an average it is slower as compared to BP in terms of computational time. The improved performance of LF-II over LF-I can be seen clearly from the figure 3.5.



(a) XOR



(b) 3-bit Parity



(c) 4-2 Encoder

Figure 3.5: Comparison of convergence time in terms of iterations between LF I and LF II

Algorithm	epochs	time(sec)	parameters
BP	2104	0.3388	$\eta = 0.5$
BP	1141	0.1848	$\eta = 0.95$
EKF	1945	2.4352	$\lambda = 0.9$
LF-I	81	0.1692	$\mu = 0.29$
LF-II	70	0.1466	$\mu = 0.3, \lambda = 0.02$

Table 3.3: comparison among three algorithms for 4-2 Encoder problem

3.4.3 4-2 Encoder

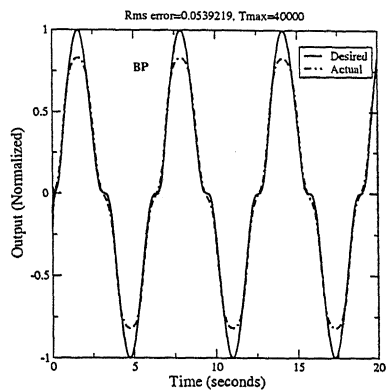
For encoder, we take a network with four inputs, two outputs and 7 hidden neurons. The simulation results are shown in the table 3.3. Here also, we find that LF-I and LF-II perform better than BP and EKF.

3.4.4 System Identification problem

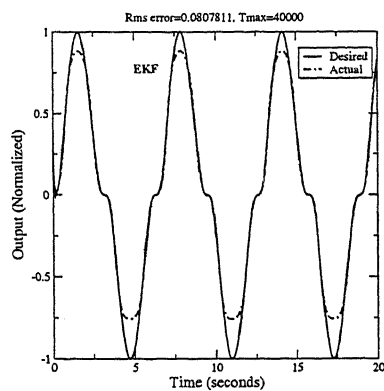
We consider the following system identification problem [19, 32]

$$x(k+1) = \frac{x(k)}{1+x^2(k)} + u^3(k) \quad (3.28)$$

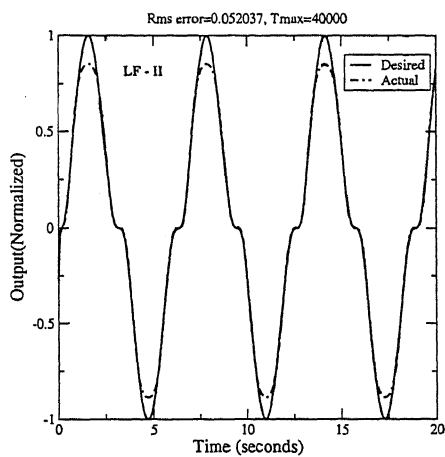
The same feedforward 2-layer network given in figure 3.1 with 7 hidden neurons is used. It has two inputs $x(k)$ and $u(k)$ respectively. BP, LF-II and EKF methods are used to train the network. The input $u(k)$ is randomly varied between 0 and 1 and 40000 training data patterns are generated. The training data are normalized between -1 and 1. After training, the network is presented with a sinusoidal input $u(k) = \sin(t)$ for three periodic cycles as test data. The neural network response and the actual model response is compared for the test data in figure 3.6. The rms error for BP, LF-II and EKF are of order 0.0539219, 0.052037, 0.0807811 respectively. The capability of EKF for system identification is



(a) BP: rms error=0.0539219



(b) EKF: rms error=0.0807811



(c) LF-II: rms error=0.052037

Figure 3.6: System identification - the trained network is tested on test-data

Algorithm	Hidden neurons	rms error/run	parameters
BP	40	0.0939937	$\eta_{1,2} = 0.1$
BP	80	0.0444109	$\eta_{1,2} = 0.1$
LF-I	40	0.0359601	$\mu = 0.4$
LF-II	40	0.0421814	$\mu = 0.4, \lambda = 0.1$

Table 3.4: Performance results for Gabor function

quite well known. LF also shows comparable and even better approximation capabilities. It should also be noted that LF-II requires very less computation time in comparison to EKF for approximation. It is also noted that if we increase the number of learning iterations, BP and EKF saturates while LF II improves its prediction.

3.4.5 2-D Gabor Function

The convolution version of complex 2-D Gabor function has the following form

$$g(x_1, x_2) = \frac{1}{2\pi\lambda\sigma^2} e^{-[(x_1/\lambda)^2 + x_2^2]/(2\sigma^2)} e^{2\pi i(u_0 x_1 + v_0 x_2)} \quad (3.29)$$

where λ is an aspect ratio, σ is a scale factor, and u_0 and v_0 are modulation parameters.

In this simulation, the following Gabor function is used.

$$g(x_1, x_2) = \frac{1}{2\pi(0.5)^2} e^{-[(x_1^2 + x_2^2)/2(0.5)^2]} \cos(2\pi(x_1 + x_2)) \quad (3.30)$$

The above function is shown in figure 3.7. We tried to train a 3-layer feedforward network to approximate this function. We started with random initial values of weights and found that none of the above three algorithms is able to converge. However, for few initial conditions, LF algorithm converge. Now, we took a radial basis function network and

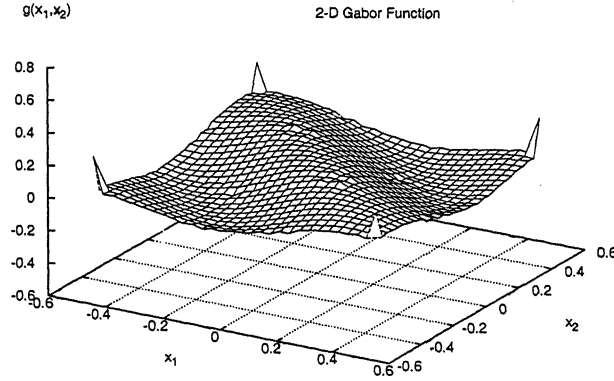


Figure 3.7: A 2-D Gabor function

applied both BP and LF algorithm for training. 5000 training data sets and 10000 test data sets were taken for this network. The rms errors were calculated for 20 different runs where each run corresponded to a different initial condition. The results are summarized in the table 3.4. This table shows that LF algorithms have better function approximation capabilities as compared to BP.

3.5 Summary

We have proposed a novel algorithm for weight update in feedforward networks using Lyapunov function approach. The key contribution of this work is to show a parallel between proposed LFI algorithm and popular BP algorithm. We showed that the fixed learning rate η in popular BP could be replaced by an adaptive learning rate $\eta_a = (\mu \frac{\|\tilde{y}\|^2}{\|J_i^T \tilde{y}\|^2})$ which can be comprehensively computed using Lyapunov function approach. Thus a

natural way to improve faster convergence of the popular BP algorithm has been demonstrated. It was also shown that a modification in Lyapunov function can lead to smooth search in the weight space. Through simulation results on three bench mark problems, we establish that proposed LF I and LF II outperform both popular BP and EKF algorithms in terms of convergence speed. We also demonstrated the dynamic system identification capabilities of the proposed algorithm using two function approximation problems and compared it with the conventional BP.

Part II

Neural Controllers

Chapter 4

Adaptive and Neural Network controllers, An analysis

4.1 Introduction

Adaptive controllers are usually used for controlling plants with uncertain parameters. These parameters are estimated and then used to calculate control input for the plant. Model reference Control and Self tuning controllers are two categories of adaptive controllers. In MRAC, the controller is so designed that the plant with controller imitates the performance of a reference model. In STC, controller is designed so that the closed loop systems gives the desired performance. An estimator which estimates the unknown parameters, is a part of the controller. Neural Network based adaptive controllers have certain advantages over conventional controllers. It can take into account not only parameter uncertainty but also unmodeled system dynamics. A comparison among these three approaches has been made for a single-link manipulator problem. More complex problems have been addressed in next chapters.

4.2 Model Reference Adaptive control of a single link manipulator

The plant model is given by

$$ml^2\ddot{q} + mgl\sin(q) = u \quad (4.1)$$

where m is mass and l is length of the manipulator arm and g is acceleration due to gravity. The equation (4.1) may be rewritten as

$$a\ddot{q} + b\sin(q) = u \quad (4.2)$$

where $a = ml^2$ and $b = mgl$ are assumed to be unknown quantities. In case, these parameters are known, it is easy to compute a control input u such that tracking objectives are achieved. The purpose of an adaptive control is to estimate the values of parameters a and b in a recursive fashion, simultaneously achieving tracking convergence.

Let's consider a sliding surface,

$$s = \dot{e} + \lambda_o e = \dot{q} - \dot{q}_r \quad (4.3)$$

$$\text{where } \dot{q}_r = \dot{q}_d - \lambda_o e$$

Choose a control input as follows:

$$u = \hat{a}\dot{q}_r - Ks + \hat{b}\sin(q) \quad (4.4)$$

where \hat{a} and \hat{b} are estimates of a and b respectively. The closed loop system is given by

$$a\ddot{q} + b\sin(q) = \hat{a}\ddot{q}_r - Ks + \hat{b}\sin(q) \quad (4.5)$$

Subtracting $a\ddot{q}_r$ from both sides of equation (4.5), we get

$$\begin{aligned} a(\ddot{q} - \ddot{q}_r) &= (\hat{a} - a)\ddot{q}_r - Ks + (\hat{b} - b)\sin(q) \\ a\dot{s} + Ks &= \tilde{a}\ddot{q}_r + \tilde{b}\sin(q) \end{aligned} \quad (4.6)$$

where $\tilde{a} = \hat{a} - a$ and $\tilde{b} = \hat{b} - b$. The response of the closed loop system can be written as

$$s = \frac{1/a}{p + k/a}(\tilde{a}\ddot{q}_r + \tilde{b}\sin(q)) = H(p)(\tilde{a}\ddot{q}_r + \tilde{b}\sin(q)) \quad (4.7)$$

where p is the laplace variable and $H(p) = \frac{1/a}{p+k/a}$. There is a basic lemma stated below which can be used to compute adaptation laws for MRAC systems [24].

Lemma 4.1 *Consider two signals e and ϕ related by the following dynamic equation*

$$e(t) = H(p)[k\phi^T(t)v(t)] \quad (4.8)$$

where $e(t)$ is a scalar output signal, $H(p)$ is a strictly positive real (SPR) transfer function, k is an unknown constant with known sign, $\phi(t)$ is a $m \times 1$ vector of function of time, and $v(t)$ is a measurable $m \times 1$ vector. If the vector ϕ varies according to

$$\dot{\phi}(t) = -\text{sgn}(k)\gamma_e v(t) \quad (4.9)$$

with γ being a positive constant, then $e(t)$ and $\phi(t)$ are globally bounded. Furthermore, if v is bounded, then

$$e(t) \rightarrow 0 \quad \text{as } t \rightarrow \infty$$

In equation (4.7), $H(p)$ is SPR. Hence, by lemma (4.1), following update law

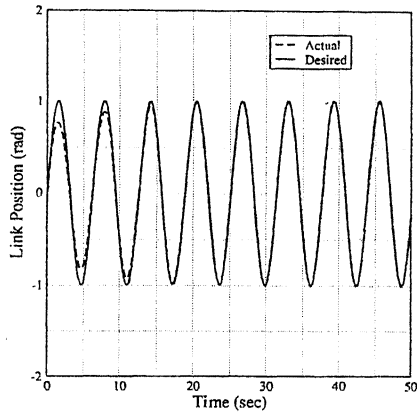
$$\dot{\hat{a}} = -\gamma s \ddot{q}_r \quad (4.10)$$

$$\dot{\hat{b}} = -\gamma s \sin(q) \quad (4.11)$$

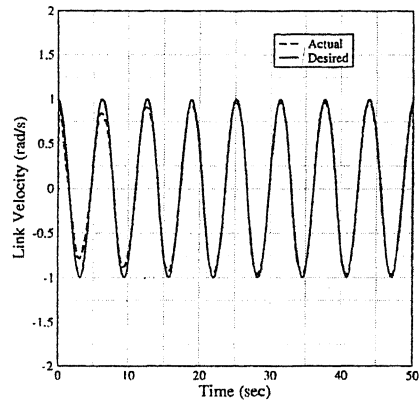
will ensure that $s \rightarrow 0$ as $t \rightarrow \infty$. The convergence of this update law can be proved by taking a suitable lyapunov function and finding its time derivative. For simulation, we have take $m = 1$, $l = 1$ and $g = 9.8$. The simulation results are shown in Fig. (4.1).

4.3 Self Tuning Control Based on Least Square Method

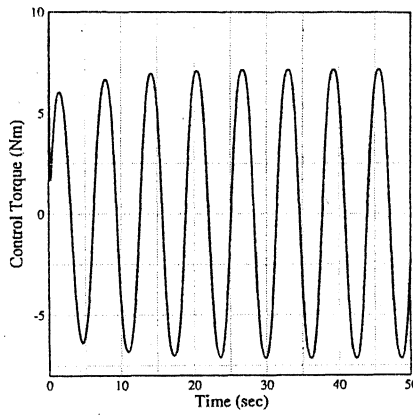
Usually, the parameters of a controller are computed from the plant parameters. In case, plant parameters are not known, it is reasonable to replace them by their estimated values as provided by a parameter estimator. A controller obtained by coupling a controller with an online (recursive) parameter estimator is called a self-tuning controller. Thus, a self-tuning controller (STC) is a controller which performs simultaneous identification of the unknown plant. Figure 4.2 illustrates the schematic structure of such an adaptive controller.



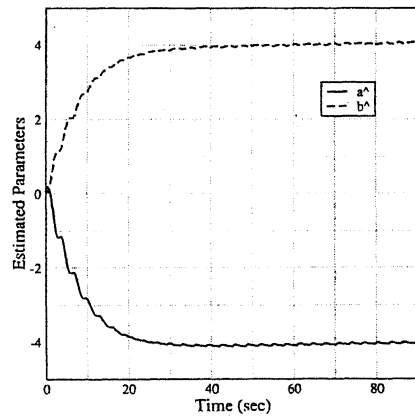
(a) Link Position



(b) Link Velocity



(c) Control input



(d) Parameter estimation

Figure 4.1: Adaptive control of Single Link Manipulator

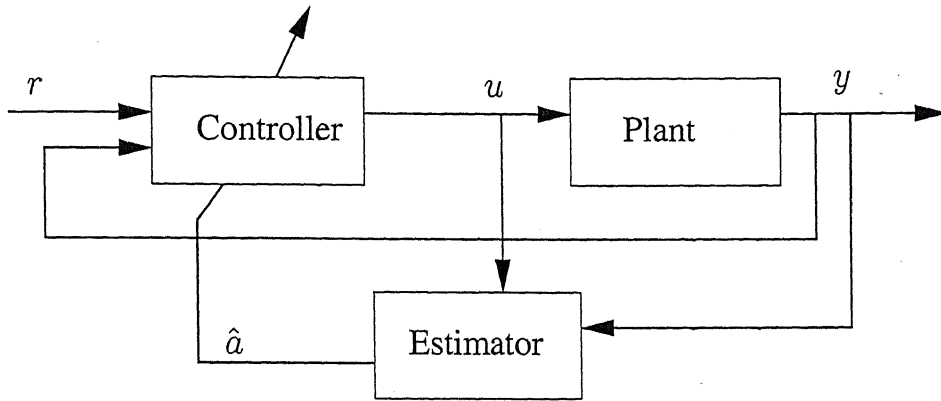


Figure 4.2: A self tuning controller

4.3.1 Parameter Estimation

When there is parameter uncertainty in a dynamic system (linear or nonlinear), one way to reduce it is to use parameter estimation, i.e, inferring the values of parameters from the measurements of input and output signals of the system [24]. Parameter estimation can be offline or online. Off-line estimation is preferable if the parameters are constant and there is sufficient time for estimation before control. In case of slowly time varying parameters, online parameter estimation is necessary to keep track of the parameter values. The essence of parameter estimation is to extract parameter information from available data concerning the system. A quite general model for parameter estimation application is in the *linear parametrization* form

$$y(t) = W(t)a \quad (4.12)$$

where the n -dimensional vector y contains the outputs of the system, the m -dimensional vector a contains unknown parameters to be estimated, and the $n \times m$ matrix $W(t)$ is a

signal matrix. It is to be noted that both y and $W(t)$ are required to be known from the measurements of the system signals.

4.3.2 Standard Least Square Estimator

In the standard least-square method[24], the estimate of the parameters is generated by minimizing the total prediction error

$$J = \int_0^t \|y(r) - W(r)\hat{a}(t)\|^2 dr \quad (4.13)$$

with respect to $\hat{a}(t)$. Since this implies the fitting of all past data, this estimate potentially has the advantage of averaging out the effects of measurement noise. The estimated parameter \hat{a} satisfies

$$[\int_0^t W^T W dr] \hat{a} = \int_0^t W^T y dr \quad (4.14)$$

which is obtained from $\frac{\partial J}{\partial \hat{a}} = 0$. Define

$$P(t) = [\int_0^t W^T(r)W(r)dr]^{-1} \quad (4.15)$$

To achieve computational efficiency, it is desirable to compute P recursively. So the above equation is replaced by a differential equation

$$\frac{d}{dt}[P^{-1}(t)] = W^T(t)W(t) \quad (4.16)$$

Differentiating equation (4.14) and using equations (4.15) and (4.16), we find that the parameter update satisfies

$$\dot{\hat{a}} = -P(t)W^T e_1 \quad (4.17)$$

where $e_1 = W\hat{a} - Wa = \hat{y} - y$ and $P(t)$ is the estimator gain matrix. By using the identity,

$$\frac{d}{dt}[PP^{-1}] = \dot{P}P^{-1} + P\frac{d}{dt}[P^{-1}] = 0$$

we obtain following weight update equation

$$\dot{P} = -PW^TWP \quad (4.18)$$

In using (4.17) and (4.18) for online estimation, one has to provide an initial parameter value and an initial gain value. $P(0)$ should be chosen as high as allowed by the noise sensitivity. \hat{a} should be initialized with some finite value.

Parameter Convergence

From (4.16), (4.17) and (4.18), one can easily show that

$$P^{-1}(t) = P^{-1}(0) + \int_0^T W^T(r)W(r)dr \quad (4.19)$$

$$\frac{d}{dt}[P^{-1}(t)\tilde{a}(t)] = 0$$

Thus,

$$\tilde{a}(t) = P(t)P^{-1}(0)\tilde{a}(0) \quad (4.20)$$

if W is such that

$$\lambda_{\min}[\int_0^t W^T W dr] \rightarrow \infty \quad \text{as} \quad t \rightarrow \infty \quad (4.21)$$

where $\lambda_{\min}[\cdot]$ denotes the smallest eigenvalue of its argument, then the gain matrix converges to zero, and the estimated parameters asymptotically converge to the true parameters. The condition (4.21) is satisfied if W is *persistently exciting* and $P \rightarrow 0$ and $\tilde{a} \rightarrow 0$.

4.3.3 Simulation Example

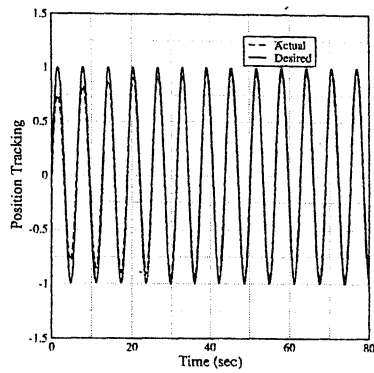
For simulation, we consider the plant (4.2) with $a = 1$ and $b = 9.8$. The model for parameter estimation is expressed as

$$\hat{u}(t) = W^T(t)\hat{z}(t) \quad (4.22)$$

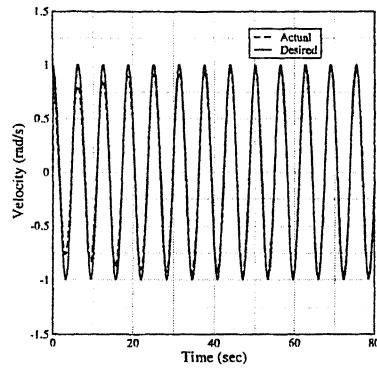
where $W(t) = [\ddot{q} \quad \sin(q)]^T$ and $\hat{z} = [\hat{a} \quad \hat{b}]^T$ and $\hat{u}(t)$ is the control input to be estimated. The desired control input is similar to (4.4) and is rewritten as

$$u(t) = \hat{a}\ddot{q}_d + \hat{b}\sin(q) + K_p e + K_d \dot{e} \quad (4.23)$$

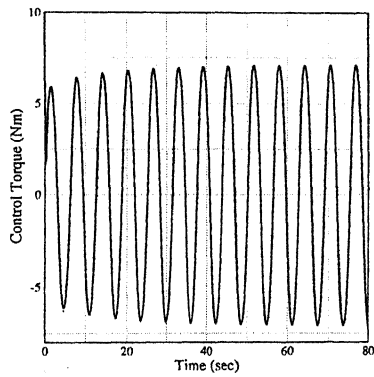
The estimation error is $e_1 = \hat{u} - u$ and the tracking error is $e = q_d - q$. The parameter vector \hat{a} is updated using (4.17) while the estimation gain matrix $P(t)$ is updated using (4.18). The simulation results are shown in figure (4.3).



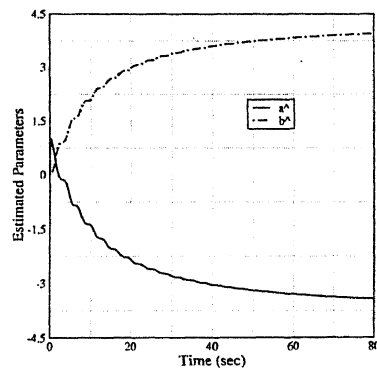
(a) Link Position



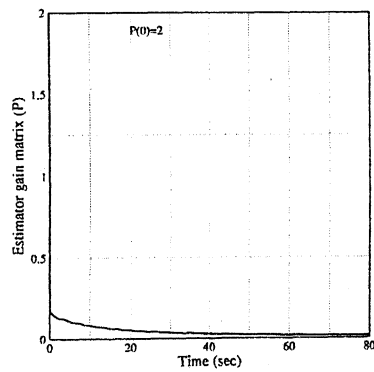
(b) Link Velocity



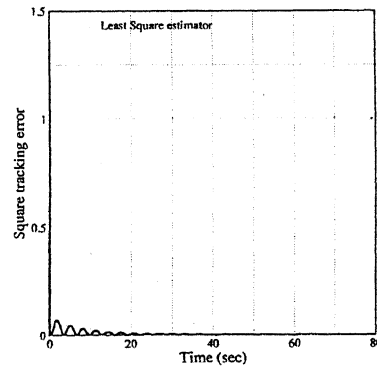
(c) Control input



(d) Parameter estimation



(e) Estimator gain matrix



(f) Tracking error

Figure 4.3: Self Tuning controller with Least Square Estimator

4.4 Neural Network based Adaptive Controller

As the number of unknown parameters increase, conventional adaptive control laws become more and more computationally intensive. By using neural networks one can get away with explicit estimation of parameters in order to compute control input for the plant. The uncertainty of parameters may be considered as unknown dynamics and a NN can be used to identify this uncertainty. Kwan et al [39, 25] have developed a weight update algorithm for Neural network that ensures the stability of closed loop system dynamics.

4.4.1 Problem definition and stability analysis

Consider the plant model (4.2) which is reproduced below for convenience

$$a\ddot{q} + b\sin(q) = u$$

Let's define $e = q_d - q$, $\dot{e} = \dot{q}_d - \dot{q}$ and $r = \dot{e} + \lambda e$, then we have

$$\begin{aligned} a\dot{r} &= a[\ddot{q}_d - \ddot{q} + \lambda\dot{e}] \\ &= a(\ddot{q}_d + \lambda\dot{e}) - a\ddot{q} \\ &= a(\ddot{q}_d + \lambda\dot{e}) + b\sin(q) - u \\ a\dot{r} &= F(\ddot{q}_d, q, e) - u \end{aligned} \tag{4.24}$$

The equation (4.28) becomes

$$\begin{aligned}\dot{V} &= -r^T K r + m \|r\| \text{tr}(\tilde{\mathbf{W}}^T \hat{\mathbf{W}}) \\ &= -r^T K r + m \|r\| \text{tr}(\tilde{\mathbf{W}}^T (\mathbf{W} - \tilde{\mathbf{W}}^r))\end{aligned}$$

By using Cauchy-Schwarz Inequality

$$\text{tr}(\tilde{\mathbf{W}}^T \mathbf{W} - \tilde{\mathbf{W}}^T \tilde{\mathbf{W}}) \leq \|\tilde{\mathbf{W}}\|_F \|\mathbf{W}\|_F - \|\tilde{\mathbf{W}}\|_F^2$$

and assuming bounded weights $\|\mathbf{W}\|_F \leq W_M$, we get

$$\begin{aligned}\dot{V} &\leq -\lambda_{\min} \|r\|^2 + m \|r\| \|\tilde{\mathbf{W}}\|_F (W_M - \|\tilde{\mathbf{W}}\|_F) \\ &\leq -\|r\| [\lambda_{\min} \|r\| + m (\|\tilde{\mathbf{W}}\|_F - \frac{W_M}{2})^2 - m \frac{W_M^2}{4}]\end{aligned}$$

where λ_{\min} is the minimum eigenvalue of K . \dot{V} is negative if the quantity inside the square bracket in the above equation is positive. This gives the condition

$$\|r\| > m \frac{W_M^2}{4\lambda_{\min}} \quad (4.30)$$

or

$$\|\tilde{\mathbf{W}}\|_F > W_M \quad (4.31)$$

Thus, \dot{V} is negative outside a compact set. The control gain K can be selected large enough so that

$$m \frac{W_M^2}{4\lambda_{\min}} < B_r$$

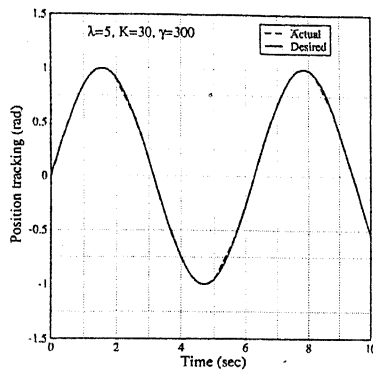
Hence, this demonstrates that both $\|r\|$ and $\|\tilde{W}\|_F$ are UUB.

4.4.2 Simulation Results

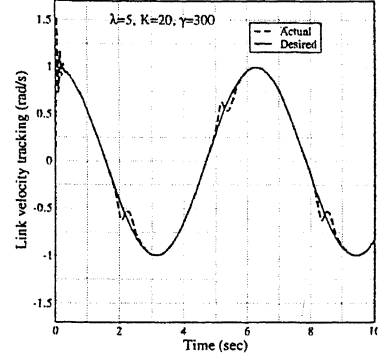
The simulation results are shown in figure (4.4). Following observations can be made from the simulation

- As can be seen in figure (4.4d), neural network is able to approximate the nonlinear function with uncertain parameters quite closely. This relieves us from deriving explicit update algorithm for each parameter which may become more and more tedious as the complexity of the problem increases.
- No restrictive assumptions like LIP (Linear in Parameters) are required for deriving the control law. So it is a more general method which can be used to control a broader class of nonlinear systems.
- Control inputs are bounded.
- For this problem, NN gives performance comparable to a high gain PD controller.

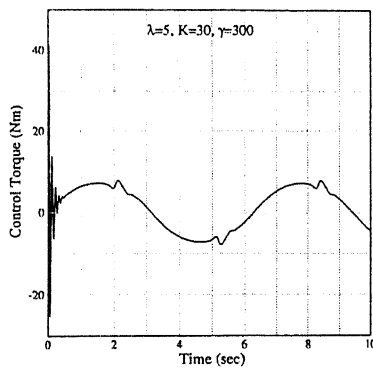
But as we will see in next chapter, NN based controller give better performance for more complex problems.



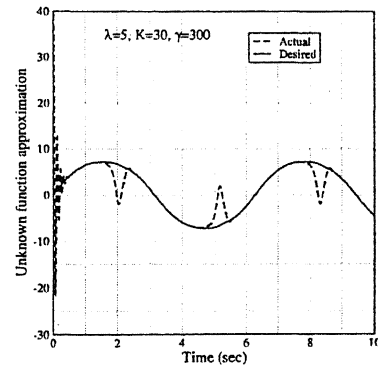
(a) Link Position



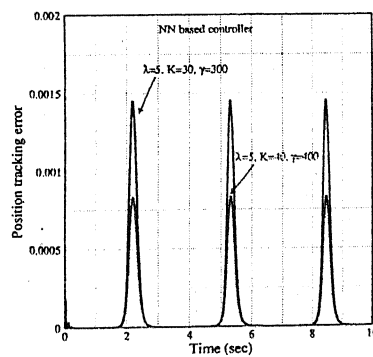
(b) Link Velocity



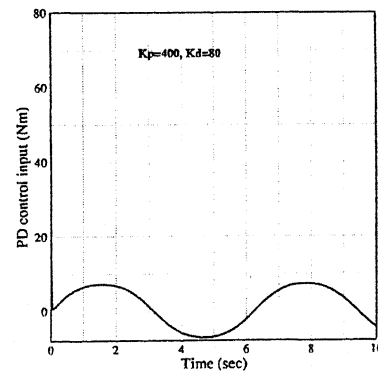
(c) Control input



(d) Function approximation



(e) Position tracking error



(f) PD control

Figure 4.4: Neural Network based Adaptive Controller for single link Manipulator

4.5 Summary

Two adaptive control techniques and one NN based control technique are used to solve a single-link manipulator problem and a comparative study has been made among them.

We found that NN removes some of the restrictive features of adaptive control and thus helps in broadening the class of problems that can be solved by adaptive controllers..

Chapter 5

Neural Network controllers for Robot Manipulators

5.1 Introduction

Most commercially available robot controllers implement some variety of PID control algorithm. As performance requirements on speed and accuracy of motion increase, PID controllers lag further behind in providing adequate robot manipulator performance. In the absence of any adaptive or learning capability, controllers might lose accuracy when uncertain parameter changes.

Robust and Adaptive controllers [15, 24] have been applied successfully to robot manipulators. A serious problem in using adaptive control in robotics is the requirement for the assumption of linearity in unknown system parameters:

$$f(x) = R(x)\xi \quad (5.1)$$

where $f(x)$ is a nonlinear robot function, $R(x)$ is a regression matrix of known robot functions and ξ is a vector of unknown parameters (e.g. masses and friction coefficients). This

is an assumption that restricts the classes of systems amenable to control. Some forms of friction, for instance, are not linear in parameters (LIP). Moreover, this LIP assumption requires one to determine the regression matrix for the system; this can involve tedious computations, and a new regression matrix must be computed for each different robot manipulator. Some of these problems can be remedied by using Neural Networks.

Neural Networks possess some very important properties, including a *universal approximation property* [40] where, for every smooth function $f(x)$, there exists a neural network such that

$$f(x) = W^T \sigma(V^T x) + \epsilon \quad (5.2)$$

for some weights W, V . This approximation holds for all x in a compact set S , and the functional estimation error ϵ is bounded so that

$$\|\epsilon\| < \epsilon_N \quad (5.3)$$

with ϵ_N a known bound dependent on S . The approximating weights may be unknown, but the NN approximation property guarantees that they exist. Unlike adaptive control, no LIP assumption is required and the property (5.2) holds for all smooth functions $f(\cdot)$.

5.2 Robot Arm Dynamics and Tracking Error Dynamics

The dynamics of rigid-link robot arms have the form :

$$M(q)\ddot{q} + V_m(q, \dot{q})\dot{q} + F(\dot{q}) + G(q) + \tau_d = \tau \quad (5.4)$$

where $M(q)$ is the inertia matrix, $V_m(q, \dot{q})$ is the Coriolis/centripetal matrix, $F(\dot{q})$ are the friction terms, $G(q)$ is the gravity vector, and $\tau_d(t)$ represents disturbances. These dynamics may also include actuators. Thus, the control input $\tau(t)$ can represent torques or motor currents, etc. The rigid dynamics have following properties:

1. **Property 5.1 (Boundedness of Inertia Matrix)** : *The inertia matrix $M(q)$ is symmetric, positive definite, and bounded so that $\mu_1 I \leq M(q) \leq \mu_2 I \quad \forall q(t)$.*
2. The Coriolis/centripetal vector $V_m(q, \dot{q})\dot{q}$ is quadratic in \dot{q} . V_m is bounded so that $\|V_m\| \leq v_B \|\dot{q}\|$, or equivalently $\|V_m\dot{q}\| \leq v_B \|\dot{q}\|^2$.
3. **Property 5.2 (Skew Symmetry)** : *The Coriolis/centripetal matrix can always be selected so that the matrix $S(q, \dot{q}) \equiv \dot{M}(q) - 2V_m(q, \dot{q})$ is skew symmetric. Therefore, $x^T Sx = 0$ for all vectors x .*
4. The gravity vector is bounded so that $\|G(q)\| \leq g_B$.
5. The disturbances are bounded so that $\|\tau_d(t)\| \leq d_B$.

The objective in this chapter is to make the robot manipulator follow a prescribed trajectory $q_d(t)$. Define the tracking error $e(t)$ and filtered tracking error $r(t)$ by

$$e = q_d - q \tag{5.5}$$

$$r = \dot{e} + \Lambda e \tag{5.6}$$

with $\Lambda > 0$, a positive definite design parameter matrix. Since (5.6) is a stable system, it follows that $e(t)$ is bounded as long as the controller guarantees that the filtered error

$r(t)$ is bounded.

Assumption 5.1 (Bounded Reference Trajectory) *The desired trajectory is bounded so that*

$$\left\| \begin{bmatrix} q_d(t) \\ \dot{q}_d(t) \\ \ddot{q}_d(t) \end{bmatrix} \right\| \leq q_B \quad (5.7)$$

with q_B a known scalar bound.

The robot dynamics can be expressed in terms of the filtered error as

$$M\dot{r} = -V_m r + f(x) + \tau_d - \tau \quad (5.8)$$

where the unknown nonlinear robot function is defined as

$$f(x) = M(q)(\ddot{q}_d + \Lambda \dot{e}) + V_m(q, \dot{q})(\dot{q}_d + \Lambda e) + F(\dot{q}) + G(q) \quad (5.9)$$

One may define, for instance,

$$x = [e^T \ \dot{e}^T \ q_d^T \ \dot{q}_d^T \ \ddot{q}_d^T]^T \quad (5.10)$$

A general sort of approximation-based controller is derived by setting

$$\tau = \hat{f} + K_v r - v(t) \quad (5.11)$$

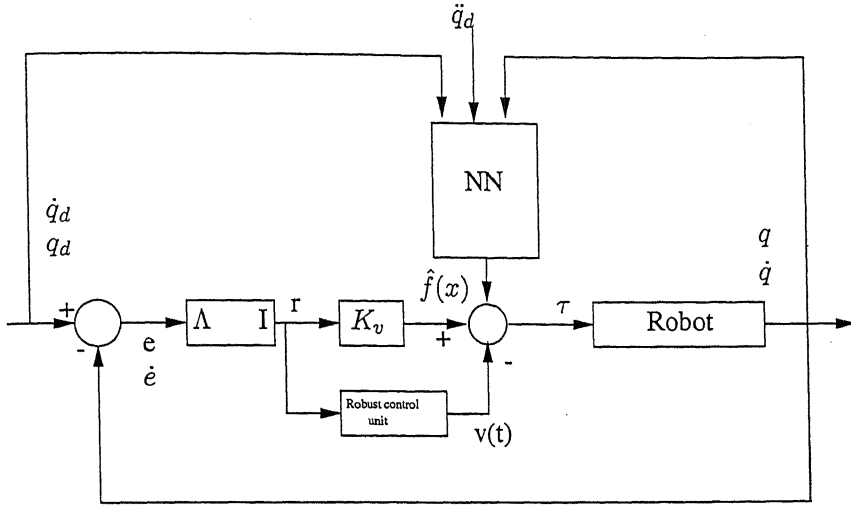


Figure 5.1: Filtered error approximation-based controller

with \hat{f} an estimate of $f(x)$, $K_v r = K_v \dot{e} + K_v \Lambda e$ an *outer PD tracking loop*, and $v(t)$ an auxiliary signal to provide robustness in the face of disturbances and modelling errors. The multiloop control structure implied by this scheme is shown in Figure 5.1. The estimate \hat{f} is obtained using NN as shown in the figure. Using this controller, the closed-loop error dynamics are

$$M\dot{r} = -V_m r - K_v r + \tilde{f} + \tau_d + v(t) \quad (5.12)$$

where the function approximation error is given by

$$\tilde{f} = f - \hat{f} \quad (5.13)$$

The following lemma is required in subsequent work.

Lemma 5.1 (Bound on NN Input x) *For each time t , $x(t)$ is bounded by*

$$\|x\| \leq c_1 + c_2\|r\| \leq q_B + c_0\|r(0)\| + c_2\|r\| \quad (5.14)$$

for computable positive constants c_0, c_1, c_2 .

The proof is available in [26]. Another assumption which is required for these kinds of NN controllers is given below.

Assumption 5.2 (Initial condition requirement) *Let the NN approximation property (5.2) hold for the function $f(x)$ given in (5.9) with a given accuracy ϵ_N for all x inside the ball of radius $b_x > q_B$. Let the initial tracking error satisfy $\|r(0)\| < (b_x - q_B)/(c_0 + c_2)$.*

This set specifies the set of allowed initial tracking errors $r(0)$. The constants q_B, c_0, c_2, b_x need not be explicitly determined. In practical situations, the initial condition requirement merely indicates that the NN should be 'large enough' in terms of the number L of hidden-layer units.

5.3 Robust Backstepping Control using Neural Networks

5.3.1 System Description

Robust control of nonlinear systems with uncertainties is of prime importance in many industrial applications. The model of many practical nonlinear systems can be expressed

in a special state-space form

$$\begin{aligned}
\dot{x}_1 &= F_1(x_1) + G_1(x_1)x_2 \\
\dot{x}_2 &= F_2(x_1, x_2) + G_2(x_1, x_2)x_3 \\
\dot{x}_3 &= F_3(x_1, x_2, x_3) + G_3(x_1, x_2, x_3)x_4 \\
&\dots = \dots \\
\dot{x}_m &= F_m(x_1, x_2, \dots, x_m) + G_m(x_1, x_2, \dots, x_m)u
\end{aligned} \tag{5.15}$$

where $x_i \in R^n$, $i = 1, 2, \dots, m$ denote the states of the system, $u \in R^n$ is the vector of control inputs. $F_i, G_i \in R^{n \times n}$, $i = 1, 2, \dots, m$ are nonlinear functions that contain both parametric and nonparametric uncertainties, and G_i 's are known and invertible. This requirement that G_i 's be invertible and known may be stringent.

The equation (5.15) is known as *strict-feedback* form [23]. The reason for this name is that the nonlinearities F_i, G_i depend only on x_1, x_2, \dots, x_i , that is, on state variables that are "fed back".

Stability of Systems [26]

Consider the following nonlinear system

$$\dot{x} = f(x, t), \quad y = h(x, t) \tag{5.16}$$

with state $x(t) \in R^n$. We say the solution is *uniformly ultimately bounded* (UUB) if there exists a compact set $U \subset R^n$ such that for all $x(t_0) = x_0 \in U$, there exists an $\epsilon > 0$ and

a number $T(\epsilon, x_0)$ such that $\|x(t)\| < \epsilon$ for all $t \geq t_0 + T$.

5.3.2 Traditional Backstepping Design

The backstepping design [23, 25] can be applied to the class of nonlinear systems (5.15) as long as the *internal dynamics are stabilizable*. In this method, first select a desirable value of x_2 , possibly a function of x_1 , denoted x_{2d} , such that in the ideal system $\dot{x}_1 = F_1(x_1, x_{2d})$, one has stable tracking by $x_1(t)$ of x_{1d} . Then in second step, select x_3 to be x_{3d} so that x_2 tracks x_{2d} and this process is repeated. Finally, select $u(t)$ such that x_m tracks x_{md} .

A number of robust and adaptive procedures exist which implement the above backstepping method. The above backstepping procedures becomes more complicated when there exist parametric uncertainties in the systems. The complications are due to the following problems with the existing robust and adaptive procedures:

1. "regression matrices" in each step of the backstepping design must be determined.

The computation of regression matrices for robot manipulators is very tedious and time consuming.

2. one basic assumption, that the unknown system parameters must satisfy the so-called "linearity-in-parameter", is quite restrictive and may not be true in many practical situations.

By using Neural Network, one can alleviate the disadvantages of the tedious and lengthy process of determining and computing regression matrices while retaining the merit of

systematic design in the backstepping control. As no LIP assumption is made, this design can be applied to a broader class of nonlinear systems.

5.3.3 Robust Backstepping controller design using NN

NN Basics

Let R denote the real numbers, R^n the real n -vectors, $R^{m \times n}$ the real $m \times n$ matrices. Let S be a compact simply connected set of R^n . With map $f : S \rightarrow R^m$, define $C^m(S)$ the functional space such that f is continuous. Define W as the collection of NN weights. Then the net output is

$$y = W^T \phi(x) \quad (5.17)$$

A general nonlinear function $f(x) \in C^m(S)$, $x(t) \in S$ can be approximated by an NN as

$$f(x) = W^T \phi(x) + \epsilon(x) \quad (5.18)$$

with $\epsilon(x)$ a NN functional reconstruction error vector. The structure of a 2-layer NN is shown in Figure 5.2. For suitable NN approximation properties, $\phi(x)$ must satisfy some conditions.

Definition 5.1 Let S be a compact simply connected set of R^n , and $\phi(x) : S \rightarrow R^{N_2}$ be integrable and bounded. Then $\phi(x)$ is said to provide a basis for $C^m(S)$ if

1. A constant function on S can be expressed as (5.18) for finite N_2 .
2. The functional range of NN (5.18) is dense in $C^m(S)$ for countable N_2 .

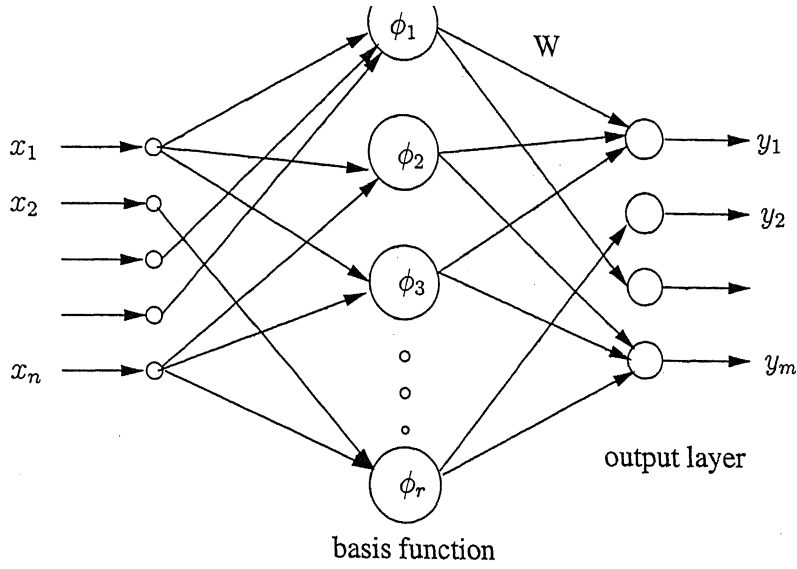


Figure 5.2: Two layer Neural Network

Here, $\phi(x)$ is chosen to be radial basis functions, as RBFs provide a universal basis for all smooth nonlinear functions.

Controller structure

Step 1 - Design fictitious controllers for x_2, x_3, \dots , and x_m : First of all, we design the fictitious controller for x_{2d} . Recalling that

$$\dot{x}_1 = F_1(x_1) + G_1(x_1)x_2 \quad (5.19)$$

Choosing the following fictitious controller

$$x_{2d} = G_1^{-1}(-\hat{F}_1 + \dot{x}_{1d} - K_1 e_1) \quad (5.20)$$

where $K_1 > 0$ is a design parameter, \hat{F}_1 is the estimate of F_1 . Substituting (5.20) into subsystem (5.19) yields the error dynamics

$$\dot{e}_1 = F_1 - \hat{F}_1 - K_1 e_1 + G_1 e_2 \quad (5.21)$$

with $e_2 = x_2 - x_{2d}$. The usual adaptive backstepping approach is to assume that the unknown parameters in F_1 are linearly parametrizable so that standard adaptive control can be used. Use of NN to approximate F_1 obviates this restriction. The next step of backstepping is to make the error $x_2 - x_{2d}$ as small as possible. Differentiating e_2 defined in (5.21) gives

$$\dot{e}_2 = \dot{x}_2 - \dot{x}_{2d} = F_2 + G_2 x_3 - \dot{x}_{2d} \quad (5.22)$$

A fictitious controller for x_3 of the form

$$x_{3d} = G_2^{-1}(-\hat{F}_2 + \dot{x}_{2d} - K_2 e_2 - G_1^T e_1) \quad (5.23)$$

can be chosen. Note that there is a coupling term $G_1 e_2$ in (5.21). The purpose of the term $G_1^T e_1$ is to compensate the effect of coupling due to $G_1 e_2$. Substituting the fictitious controller (5.23) into (5.22) gives

$$\dot{e}_2 = F_2 - \hat{F}_2 - K_2 e_2 - G_1^T e_1 + G_2 e_3 \quad (5.24)$$

with $e_3 = x_3 - x_{3d}$, $K_2 > 0$ a design parameter and \hat{F}_2 the estimate of F_2 . In a similar fashion, we can design a fictitious controller for x_m to make the error $e_{m-1} =$

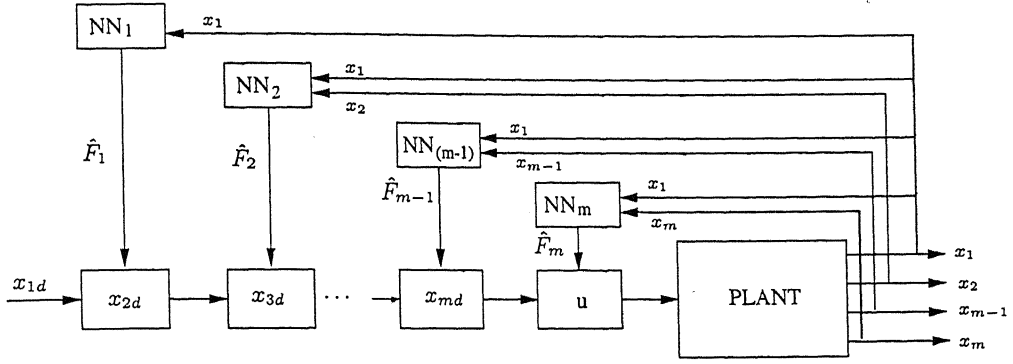


Figure 5.3: Backstepping NN control of nonlinear systems in “strict-feedback” form

$x_{m-1} - x_{(m-1)d}$ as small as possible, i.e.,

$$x_{md} = G_{m-1}^{-1}(-\hat{F}_{m-1} + \dot{x}_{(m-1)d} - K_{m-1}e_{m-1} - G_{m-2}^T e_{m-2}) \quad (5.25)$$

The dynamics of $e_{m-1} = x_{m-1} - x_{(m-1)d}$ is then governed by

$$\dot{e}_{m-1} = F_{m-1} - \hat{F}_{m-1} - K_{m-1}e_{m-1} - G_{m-2}^T e_{m-2} + G_{m-1}e_m \quad (5.26)$$

with $e_m = x_m - x_{md}$, $K_{m-1} > 0$ a design parameter and \hat{F}_{m-1} the estimate of F_{m-1} . Here, NNs are used to approximate the complicated nonlinear functions F_i 's, $i = 1, 2, \dots, m$. As a result, no regression matrices are needed and the controller is re-usable for different systems within the same class of nonlinear systems.

Step 2 - Design of Actual control u : Differentiating $e_m = x_m - x_{md}$ defined in (5.26)

yields

$$\dot{e}_m = \dot{x}_m - \dot{x}_{md} = F_m + G_m u - \dot{x}_{md} \quad (5.27)$$

Choosing the controller of the form

$$u = G_m^{-1}(-\hat{F}_m + \dot{x}_{md} - K_m e_m - G_{m-1}^T e_{m-1}) \quad (5.28)$$

gives the following dynamics for error e_m ,

$$\dot{e}_m = F_m - \hat{F}_m - K_m e_m - G_{m-1}^T e_{m-1} \quad (5.29)$$

with $K_m > 0$ a design parameter and \hat{F}_m the estimate of F_m . The overall control structure is shown in Figure 5.3.

Bounding assumptions, Error dynamics and Weight tuning algorithm

Assume that the nonlinear functions F_i 's, $i = 1, 2, \dots, m$ in equations (5.21), (5.24), (5.26), and (5.29) can be represented by m 2-layer neural nets for some constant "ideal" weights W_i , $i = 1, 2, \dots, m$, i.e.,

$$F_i = W_i^T \phi_i + \epsilon_i, \quad \|\epsilon_i\| < \epsilon_{iN} = \text{constant} \quad (5.30)$$

for $i = 1, 2, \dots, m$, where ϕ_i 's provide suitable basis functions for m NNs. The net reconstruction errors ϵ_i 's are bounded by known constants ϵ_{iN} , $i = 1, 2, \dots, m$. Define the NN functional estimate of F_i in (5.30) by

$$\hat{F}_i = \hat{W}_i^T \phi_i, \quad i = 1, 2, \dots, m \quad (5.31)$$

with \hat{W}_i the current NN weight estimates provided by the tuning algorithm. Then the error dynamics (5.21), (5.24), (5.26), (5.29) become

$$\begin{aligned}
\dot{e}_1 &= \tilde{W}_1^T \phi_1 - K_1 e_1 + G_1 e_2 + \epsilon_1 \\
\dot{e}_2 &= \tilde{W}_2^T \phi_2 - K_2 e_2 - G_1^T e_1 + G_2 e_3 + \epsilon_2 \\
\dot{e}_3 &= \tilde{W}_3^T \phi_3 - K_3 e_3 - G_2^T e_2 + G_3 e_4 + \epsilon_3 \\
&\dots = \dots \\
\dot{e}_m &= \tilde{W}_m^T \phi_m - K_m e_m - G_{m-1}^T e_{m-1} + \epsilon_m
\end{aligned} \tag{5.32}$$

Define $\zeta = [e_1^T \ e_2^T \ \dots \ e_m^T]^T$, $\epsilon = [\epsilon_1^T \ \epsilon_2^T \ \dots \ \epsilon_m^T]^T$, $\tilde{Z} = \text{diag}\{\tilde{W}_1, \tilde{W}_2, \dots, \tilde{W}_m\}$, $\Gamma = \text{diag}\{\Gamma_1, \Gamma_2, \dots, \Gamma_m\}$, $K = \text{diag}\{K_1, K_2, \dots, K_m\}$, $\phi = [\phi_1^T \ \phi_2^T \ \dots \ \phi_m^T]^T$ and

$$H = \begin{bmatrix} 0 & G_1 & 0 & \dots & 0 \\ -G_1^T & 0 & G_2 & \dots & 0 \\ 0 & -G_2^T & 0 & \dots & \dots \\ \dots & \dots & \dots & \dots & G_{m-1} \\ 0 & 0 & \dots & -G_{m-1}^T & 0 \end{bmatrix}$$

The error dynamics (5.32) can be expressed in terms of the above quantities as

$$\dot{\zeta} = -K\zeta + \tilde{Z}^T \phi + H\zeta + \epsilon \tag{5.33}$$

Note that the term $H\zeta$ denotes the couplings between the error dynamics (5.33). The matrix H is skew-symmetric. Along with (5.1), another assumption which is quite common

in the neural network literature is stated next.

Assumption 5.3 *The ideal weights are bounded by known positive values so that*

$$\|W_i\|_F \leq W_{iM}, \quad i = 1, 2, \dots, m \quad (5.34)$$

or equivalently, $\|Z\| \leq Z_m$, where Z_m is known. The symbol $\|\cdot\|_F$ denotes the Frobenius norm.

Kwan et al. [25, 39] proposed a robust weight tuning algorithm for the 2-layer NN which is given below.

Theorem 5.1 (Weight tuning algorithm) *Suppose Assumptions (5.1) and (5.3) are satisfied. Take the control input (5.28) with NN weight tuning be provide by*

$$\dot{\hat{W}}_i = \Gamma_i \phi_i e_i^T - k_w \Gamma_i \|\zeta\| \hat{W}_i, \quad i = 1, 2, \dots, m \quad (5.35)$$

with constant matrices $\Gamma_i = \Gamma_i^T > 0, i = 1, 2, \dots, m$, and scalar positive constant k_w .

Then the errors $e_i(t), i = 1, 2, \dots, m$, and NN weights are UUB.

The errors $e_i(t), i = 1, 2, \dots, m$, can be kept as small as possible by increasing gains K in (5.33). The proof has already been discussed in the last chapter. This tuning algorithm can be implemented online and does not require any offline training.

5.4 Singular Perturbation Design

5.4.1 Introduction

The method of singular perturbations [26] recognizes the fact that a large class of systems have slow dynamics and fast dynamics which operate on a very different time-scales and are essentially independent. The control input can therefore be decomposed into a fast portion and a slow portion, which has the effect of doubling the number of control inputs. This is accomplished by a *time-scale separation* that allows one to split the dynamics into a *slow subsystem* and a *fast subsystem*.

5.4.2 Singular Perturbations for Nonlinear Systems

A large class of nonlinear systems can be described by the equations

$$\dot{x}_1 = f_1(x, u) \quad (5.36)$$

$$\varepsilon \dot{x}_2 = f_{21}(x_1) + f_{22}(x_1)x_2 + g_2(x_1)u \quad (5.37)$$

where the state $x = [x_1^T \ x_2^T]^T$ is decomposed into two portions. The left-hand side of the second equation is premultiplied by $\varepsilon \ll 1$, indicating that the dynamics of x_2 are much faster than those of x_1 . In fact, the variable x_1 develops on the standard time-scale t , while $\frac{d}{dT}x_2 = \varepsilon \dot{x}_2$ with

$$T = \frac{t}{\varepsilon} \quad (5.38)$$

so that the natural time-scale for x_2 is a faster one defined by T . The control variable $u(t)$ is required to manipulate both x_1 and x_2 . Using the technique of singular perturbations, its *effectiveness* can be increased under certain conditions as explained next. This allows simplified control design.

Slow/Fast Subsystem Decomposition

The system may be decomposed into a slow subsystem and a fast subsystem to increase the control effectiveness. To accomplish this, define all variables to consist of a *slow part*, denoted by overbar, and a *fast part*, denoted by tilde. Thus,

$$x_1 = \bar{x}_1 + \tilde{x}_1, \quad x_2 = \bar{x}_2 + \tilde{x}_2, \quad u = \bar{u} + \tilde{u} \quad (5.39)$$

To derive the slow dynamics, set $\varepsilon = 0$ and replace all variables by their slow portions.

From (5.37) one obtains

$$0 = f_{21}(\bar{x}_1) + f_{22}(\bar{x}_1)\bar{x}_2 + g_2(\bar{x}_1)\bar{u}$$

which may be solved to obtain

$$\bar{x}_2 = -f_{22}^{-1}(\bar{x}_1)[f_{21}(\bar{x}_1) + g_2(\bar{x}_1)\bar{u}] \quad (5.40)$$

under the condition that $f_{22}(x_1)$ is *invertible*. This is known as the *slow manifold* equation, and reveals that the slow portion of x_2 is dependent completely on the slow portion of x_1

and the slow portion of the control input. Now one obtains from (5.36) the *slow subsystem* equation

$$\dot{\bar{x}}_1 = f_1(\bar{x}_1, \bar{x}_2, \bar{u}) \quad (5.41)$$

with \bar{x}_2 given by the slow manifold equation. One can notice that this equation is of *reduced order*. To study the fast dynamics, one works in the time scale T , assuming that the slow variables vary negligibly in this time scale. From (5.36) one has

$$\begin{aligned} \dot{x}_1 &= \frac{d}{dt}x_1 = \frac{1}{\varepsilon} \frac{d}{dT}x_1 = f_1(x, u) \\ \frac{d}{dT}x_1 &= \frac{d}{dT}(\bar{x}_1 + \tilde{x}_1) = \frac{d}{dT}\tilde{x}_1 = \varepsilon f_1(x, u) \approx 0 \end{aligned}$$

Now, one may write from (5.37)

$$\varepsilon \frac{d}{dt}x_2 = \frac{d}{dT}(\bar{x}_2 + \tilde{x}_2) = f_{21}(\bar{x}_1) + f_{22}(\bar{x}_1)(\bar{x}_2 + \tilde{x}_2) + g_2(\bar{x}_1)(\bar{u} + \tilde{u}),$$

whence substitution from (5.40) and the assumption that $\frac{d}{dT}\bar{x}_2 = 0$ yields

$$\frac{d}{dT}\tilde{x}_2 = f_{22}(\bar{x}_1)\tilde{x}_2 + g_2(\bar{x}_1)\tilde{u} \quad (5.42)$$

This is a *fast subsystem*, which is *linear* since \bar{x}_1 is constant in the T time scale.

Composite controls Design and Tikhonov's Theorem

The singular perturbation slow/fast decomposition suggests the following control design technique. Design a slow control \bar{u} for the slow subsystem (5.41) using any method.

Design a fast control \tilde{u} for the fast subsystem (5.42) using any method. Then, apply the *composite control*

$$u = \bar{u} + \tilde{u} \quad (5.43)$$

Using this technique, one performs *two control designs*, and effectively obtains two independent control input components that are simply added to produce $u(t)$. This independent control design of two control inputs practically increases the control effectiveness. The composite control approach works due to an extension of *Tikhonov's Theorem*, which also relates the slow/fast decomposition (5.41)-(5.42) to the original system description (5.36)-(5.37). It states that if $f_{22}(x_1)$ is invertible and the linear system (5.42) is *stabilizable* considering \bar{x}_1 as slowly time-varying (i.e. practically constant), then one has

$$x_1 = \bar{x}_1 + O(\varepsilon) \quad (5.44)$$

$$x_2 = \bar{x}_2 + \tilde{x}_2 + O(\varepsilon) \quad (5.45)$$

where $O(\varepsilon)$ denotes terms of order ε . From these notions one obtains the concept of \bar{x}_2 as a *boundary-layer correction* term due to the fast dynamics, and of \tilde{u} as a *boundary-layer correction control* term that manages the high-frequency (fast) motion.

5.5 Applications

5.5.1 Rigid Link Electrically Driven Robot Manipulator

For simplicity, we assume the actuator is a permanent magnet dc motor. The model for an n -link RLED robot is given by

$$M(q)\ddot{q} + V_m(q, \dot{q})\dot{q} + G(q) + F(\dot{q}) + T_L = K_T I \quad (5.46)$$

$$L\dot{I} + R(I, \dot{q}) + T_E = u_E \quad (5.47)$$

with $q, \dot{q}, \ddot{q} \in R^n$ denoting link position, velocity, and acceleration vectors, respectively, $M(q) \in R^{n \times n}$ the inertia matrix, $V_m(q, \dot{q}) \in R^n$ the Centripetal-Coriolis matrix, $G(q) \in R^n$ the gravity vector, $F(\dot{q}) \in R^n$ representing the friction terms, $T_L \in R^n$ the additive bounded disturbance, $I \in R^n$ the armature current, $K_T \in R^{n \times n}$ the positive definite constant diagonal matrix which characterizes the electro-mechanical conversion between current and torque, $L \in R^{n \times n}$ a positive definite constant diagonal matrix denoting the electrical inductance, $R(I, \dot{q}) \in R^{n \times n}$ representing the electrical resistance and the motor back-electromotive force, $u_E \in R^n$ the control vector representing the motor terminal voltages, and $T_E \in R^n$ representing an additive bounded voltage disturbance.

The rigid dynamics properties are enumerated in the section 5.2. The torque transmission matrix K_T is assumed to be bounded by

$$k_1 \|x\|^2 \leq x^T K_T x \leq k_2 \|x\|^2, \text{ for arbitrary vector } x \quad (5.48)$$

where k_1 and k_2 are positive scalar bounding constants. The inductance matrix L is also assumed to be bounded by

$$l_1 \|x\|^2 \leq x^T L x \leq l_2 \|x\|^2, \text{ for arbitrary vector } x \quad (5.49)$$

where l_1 and l_2 are positive scalar bounding constants.

5.5.2 Control Objective and Central Ideas of NN RLED Controller Design

The control objective is to develop a link position tracking controller [39] for the RLED robot dynamics given by (5.46) based on inexact knowledge of manipulator dynamics. Following the discussion similar to that in section 5.2, the robot dynamics can be expressed in terms of filtered error as

$$M\dot{r} = F_1 - V_m r + T_L - K_T I \quad (5.50)$$

where $\Lambda \in R^{n \times n}$ is a positive definite control gain and the complicated nonlinear function F_1 is defined as

$$F_1 = M(q)(\ddot{q}_d + \Lambda \dot{e}) + V_m(q, \dot{q})(\dot{q}_d + \Lambda e) + G(q) + F(\dot{q}) \quad (5.51)$$

Design Steps

Step 1: Treat current I as a fictitious control signal to the error dynamics ,

Then (5.50) can be rewritten as

$$M\dot{r} = F_1 - V_m r + T_L - K_T I_d + K_T \eta \quad (5.52)$$

where $\eta = I_d - I$ is an error signal which is to be minimized in the second step. The control objective of the first step is to design an NN controller for I_d to make r as small as possible. The fictitious controller can be selected as

$$I_d = \frac{1}{k_1} [\hat{F}_1 + k_\tau r + \nu_\tau] \quad (5.53)$$

where $\hat{F}_1 = \hat{W}_1^T \phi_1$, k_τ is a positive scalar, ν_τ is a robustifying term to be defined shortly, and k_1 is defined in (5.48). Substituting (5.53) into (5.52) gives

$$\begin{aligned} M\dot{r} = & -V_m r + \tilde{W}_1^T \phi_1 - k_\tau \frac{K_T}{k_1} r + \varepsilon_1 + T_L \\ & + \left(I - \frac{K_T}{k_1} \right) \hat{W}_1^T \phi_1 - \frac{K_T}{k_1} \nu_\tau + K_T \eta \end{aligned} \quad (5.54)$$

As one can see that there is an unknown term $\left(I - \frac{K_T}{k_1} \right) \hat{W}_1^T \phi_1$ in (5.54) because K_T is usually unknown. The role of the robustifying term ν_τ is to suppress the effect of this signal. The form of ν_τ is chosen to be

$$\nu_\tau = \rho_\tau \text{sgn}(\tau) \quad (5.55)$$

where

$$\rho_\tau = \|\hat{W}_1^T \phi_1\| b_k \quad (5.56)$$

and b_k in (5.56) stand for the upper bound of $\|(I - \frac{K_T}{k_1})\|$.

Step 2: The second step is to design a second NN controller for u_E such that the error signal η is as small as possible. Differentiating $\eta = I_d - I$, using (5.47) and multiplying L on both sides of the resulting expression yields

$$L\dot{\eta} = F_2 + T_E - u_E \quad (5.57)$$

where F_2 is very complicated nonlinear function of q, \dot{q}, r, I_d , and I . The control signal u_E can be chosen as

$$u_E = \hat{F}_2 + k_\nu \eta \quad (5.58)$$

where $\hat{F}_2 = W_2^T \phi_2$ and $k_\nu > 0$. Inserting (5.58) into (5.57) gives

$$L\dot{\eta} = \tilde{W}_2^T \phi_2 + \varepsilon_2 + T_E - k_\nu \eta \quad (5.59)$$

The controller structure is shown in Fig. 5.4.

Step 3: Weight update algorithm and stability analysis

The weight tuning algorithm as stated in theorem 5.1 is used here. It is shown that all errors such as weight updates and tracking errors are UUB. The weight tuning algorithm

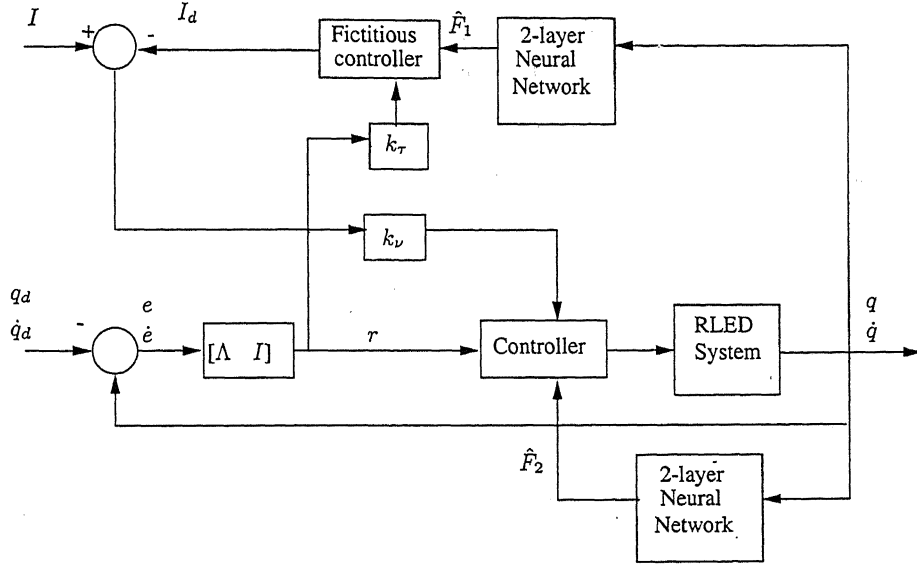


Figure 5.4: NN controller structure for RLED

for two networks is restated below for convenience.

$$\dot{\hat{W}}_1 = \Gamma_1 \phi_1 r^T - k_w \Gamma_1 \|\zeta\| \hat{W}_1 \quad (5.60)$$

$$\dot{\hat{W}}_2 = \Gamma_2 \phi_2 r^T - k_w \Gamma_2 \|\zeta\| \hat{W}_2 \quad (5.61)$$

with any constant matrices $\Gamma_1 = \Gamma_1^T > 0$, $\Gamma_2 = \Gamma_2^T > 0$, $\zeta = [r^T \ \eta^T]^T$, and scalar positive constant k_w .

Simulation

The model for RLED is described in the form (5.46) with

$$M(q) = \begin{bmatrix} a + b \cos(q_2) & c + \frac{b}{2} \cos(q_2) \\ c + \frac{b}{2} \cos(q_2) & c \end{bmatrix}$$

$$V_m \dot{q} = \begin{bmatrix} -b \sin(q_2)(\dot{q}_1 \dot{q}_2 + 0.5 \dot{q}_2^2) \\ 0.5b \sin(q_2) \dot{q}_1^2 \end{bmatrix},$$

$$G(q) = \begin{bmatrix} d \cos(q_1) + e \cos(q_2) \\ e \cos(q_1 + q_2) \end{bmatrix}$$

$$a = l_2^2 m_2 + l_1^2 (m_1 + m_2), \quad b = 2l_1 l_2 m_2,$$

$$c = l_2^2 m_2, \quad d = (m_1 + m_2) l_1 g_0, \quad e = m_2 l_2 g_0$$

The parameter values are $l_1 = 1m$, $l_2 = 1m$, $m_1 = 0.8Kg$, $m_2 = 2.3Kg$, and $g_0 = 9.8m/s^2$. The actuator dynamics are assumed to be the permanent magnet direct-current motor (5.47). The parameters of motor are $R_j = 1\Omega$, $L_j = 0.01H$, $K_j^T = 2.0Nm/A$, $j = 1, 2$. The desired joint trajectories are defined as $q_{d1}(t) = \sin(t)$, $q_{d2}(t) = \cos(t)$. The inputs to the NNs are given by

$$x = [\zeta_1^T \quad \zeta_2^T \quad \cos(q)^T \quad \sin(q)^T \quad I^T \mathbf{1}]^T$$

where $\zeta_1 = \ddot{q}_d + \Lambda \dot{e}$ and $\zeta_2 = \dot{q}_d + \Lambda e$. Figure 5.5 shows the simulation results with PD control with $K = \text{diag}\{100, 100\}$ and $K_d = \text{diag}\{20, 20\}$. Figure 5.6 shows the simulation results for NN back-stepping control with $k_\tau = \text{diag}\{60, 60\}$ and $k_\nu = \text{diag}\{1.5, 1.5\}$.

Remarks:

- The problem of weight initialization occurring in other approaches in the literature

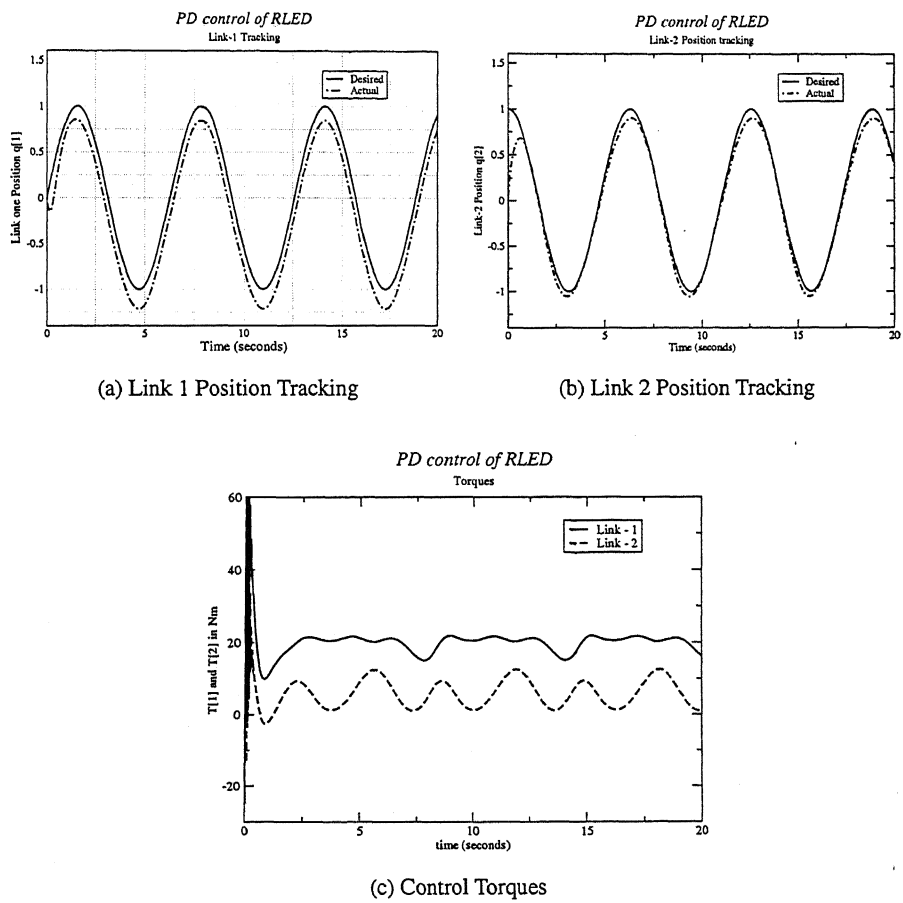


Figure 5.5: PD control of 2-link RLED

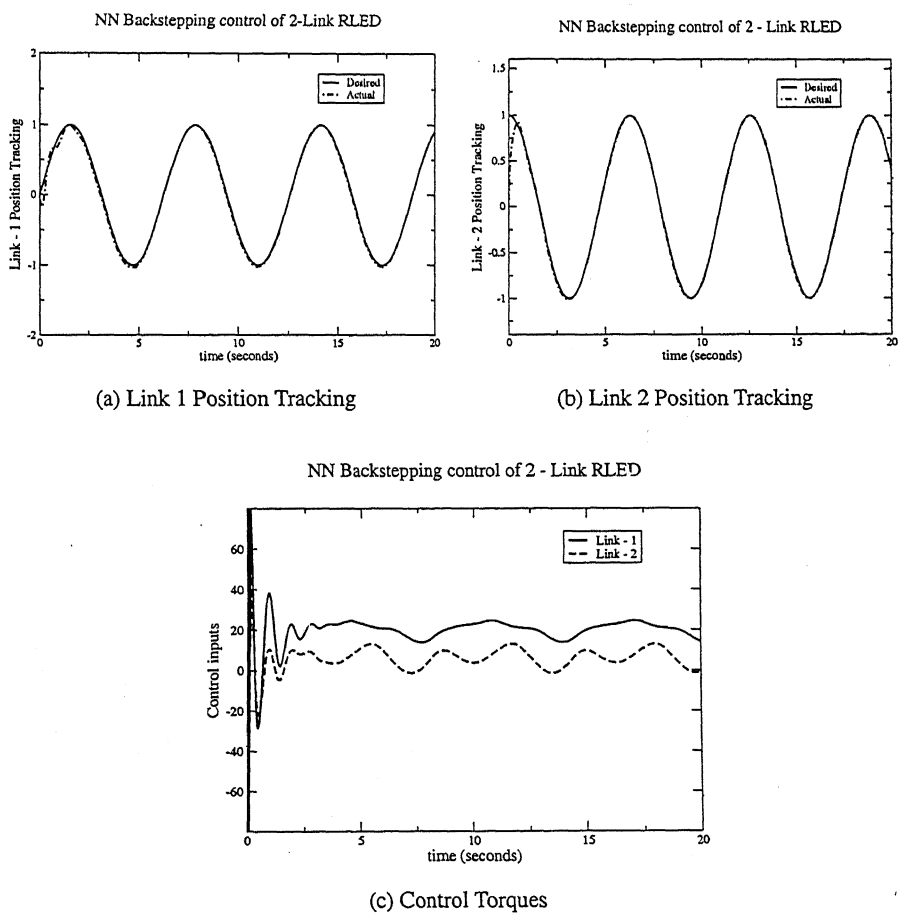


Figure 5.6: NN Back Stepping control of 2-link RLED

does not arise, since weights $\hat{W}_i(0)$ are taken as zeros.

- The tuning algorithm guarantees the boundedness of weights \hat{W}_i 's.
- In PD control one can get better tracking performance and thus smaller tracking error by increasing the gains. However, it is a well known fact that high-gain feedback may not be desirable since it may excite some high-frequency unmodeled dynamics.
- The NN controller can indeed improve the tracking performance without resorting to high-gain feedback.

5.5.3 Flexible Link Manipulator

Flexible-link robotic systems [26] comprise an important class of systems that include lightweight arms for assembly, civil infrastructure bridge/vehicle systems, military tank gun barrel applications, and large-scale space structures. Their key feature is the presence of *vibratory modes* that tend to disturb or even destabilize the motion. They are described as infinite-dimensional dynamical systems using partial differential equations (PDE); some assumptions make them tractable by allowing one to describe them using ordinary differential equation (ODE), which is finite-dimensional. The dynamics of flexible link manipulator can be described as

$$M(q)\ddot{q} + V_m(q, \dot{q})\dot{q} + Kq + F(\dot{q}) + G(q) + \tau_d = B(q)\tau \quad (5.62)$$

where $q(t) \in R^n$ consists of all rigid and flexible mode variables. One may partition the dynamics (5.62) according to the rigid and flexible modes as

$$\begin{bmatrix} M_{rr} & M_{rf} \\ M_{fr} & M_{ff} \end{bmatrix} \begin{bmatrix} \ddot{q}_r \\ \ddot{q}_f \end{bmatrix} + \begin{bmatrix} V_{rr} & V_{rf} \\ V_{fr} & V_{ff} \end{bmatrix} \begin{bmatrix} \dot{q}_r \\ \dot{q}_f \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & K_{ff} \end{bmatrix} \begin{bmatrix} q_r \\ q_f \end{bmatrix} + \begin{bmatrix} F_r \\ 0 \end{bmatrix} + \begin{bmatrix} G_r \\ 0 \end{bmatrix} = \begin{bmatrix} B_r \\ B_f \end{bmatrix} \tau \quad (5.63)$$

The major difference between (5.62) and rigid robot equation (5.46) is that the input matrix $B(q)$ has more rows than columns, so that the flexible-link arms have *reduced control effectiveness* and the techniques used for rigid robots cannot be directly applied. An ameliorating factor is that, like the rigid-link case, flexible-link dynamics satisfy the properties enumerated in section 5.2. In equation (5.62) one can notice that gravity and friction only act on the rigid modes, while the flexibility effects in K only effect the flexible modes. The control matrix B_r is nonsingular. The equation (5.63) may also be written as

$$\begin{aligned} M_{rr}\ddot{q}_r + M_{rf}\ddot{q}_f + V_{rr}\dot{q}_r + V_{rf}\dot{q}_f + F_r(\dot{q}_r) + G_r(q_r) &= B_r\tau \\ M_{fr}\ddot{q}_r + M_{ff}\ddot{q}_f + V_{fr}\dot{q}_r + V_{ff}\dot{q}_f + K_{ff}q_f &= B_f\tau \end{aligned} \quad (5.64)$$

To write a state equation, define

$$\begin{bmatrix} H_{rr} & H_{rf} \\ H_{fr} & H_{ff} \end{bmatrix} = \begin{bmatrix} M_{rr} & M_{rf} \\ M_{fr} & M_{ff} \end{bmatrix}^{-1} \quad (5.65)$$

Multiply (5.63) by (5.65) from the left, rearrange the terms, and write

$$\ddot{q}_r = -V_{rr}^1 \dot{q}_r - V_{rf}^1 \dot{q}_f - K_{rf}^1 q_f - F_r^1 - G_r^1 + B_r^1 \tau \quad (5.66)$$

$$\ddot{q}_f = -V_{fr}^1 \dot{q}_r - V_{ff}^1 \dot{q}_f - K_{ff}^1 q_f - F_f^1 - G_f^1 + B_f^1 \tau \quad (5.67)$$

with:

$$V_{rr}^1 = H_{rr} V_{rr} + H_{rf} V_{fr}, \quad V_{fr}^1 = H_{fr} V_{rr} + H_{ff} V_{fr},$$

$$V_{rf}^1 = H_{rr} V_{rf} + H_{rf} V_{ff}, \quad V_{ff}^1 = H_{fr} V_{rf} + H_{ff} V_{ff},$$

$$K_{rf}^1 = H_{rf} K_{ff}, \quad K_{ff}^1 = H_{ff} K_{ff},$$

$$F_r^1 = H_{rr} F_r, \quad F_f^1 = H_{fr} F_r,$$

$$G_r^1 = H_{rr} G_r, \quad G_f^1 = H_{fr} G_r,$$

$$B_r^1 = H_{rr} B_r + H_{rf} B_f, \quad B_f^1 = H_{fr} B_r + H_{ff} B_f.$$

Now, equations (5.66)-(5.67) can be placed into state-space form $\dot{x} = f(x, u)$ by defining

the state, for instance, as $x = [q \dot{q}]^T = [q_r \ q_f \ \dot{q}_r \ \dot{q}_f]^T$.

Control Difficulties :

The objective is basically to force the rigid mode variable $q_r \in R^{n_r}$ to follow desired trajectories. The control input available is $\tau \in R^{n_r}$, since there is one actuator per link. However, the extra state $q_f \in R^{n_f}$ introduces n_f *additional vibratory degrees of freedom* that require control to suppress the vibrations. Therefore, the number of inputs is less than the number of degrees of freedom. Moreover, it turns out that by selecting the control input $\tau(t)$ to achieve practical tracking performance of rigid variable $q_r(t)$, one actually destabilizes the flexible modes $q_f(t)$. This is due to the *non-minimum phase nature* of zero dynamics of flexible-link robot arms. Here, singular perturbation technique [26] is used to solve this control problem for a one-link robot arm with two retained flexible modes.

Open-Loop Behaviour of Flexible-link Robot Arm

The model of one-link robot arm with pinned-pinned boundary conditions and two retained modes is given by (5.62) with

$$M = \begin{bmatrix} 2.2024 & 0.0517 & 0.0410 \\ 0.0517 & 0.0026 & 0.0036 \\ 0.0410 & 0.0036 & 0.0080 \end{bmatrix}, \quad V_m = \begin{bmatrix} 0.0200 & 0.0013 & 0.0027 \\ 0.0013 & 0.0001 & 0.0002 \\ 0.0027 & 0.0002 & 0.0004 \end{bmatrix}$$

$$K = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 14.0733 & 0 \\ 0 & 0 & 225.1734 \end{bmatrix}, \quad G = 0, \quad B = \begin{bmatrix} 1.0000 \\ 0.0668 \\ 0.0668 \end{bmatrix}$$

The open-loop poles are found to be

$$s = \begin{pmatrix} 0 \\ -0.01 \\ -0.0002 \pm 89.97i \\ -0.0000 \pm 342.01i \end{pmatrix}$$

There are two poles near $s = 0$ corresponding to rigid joint angle q_r and two complex pole pair, almost completely undamped, one corresponding to first mode with frequency 14.32 Hz and one corresponding to second mode with frequency 54.43 Hz. The open-loop response of flexible arm is shown in Fig. 5.7. The torque profile chosen for the open-loop response is also shown in the same figure.

NN based singular perturbation design

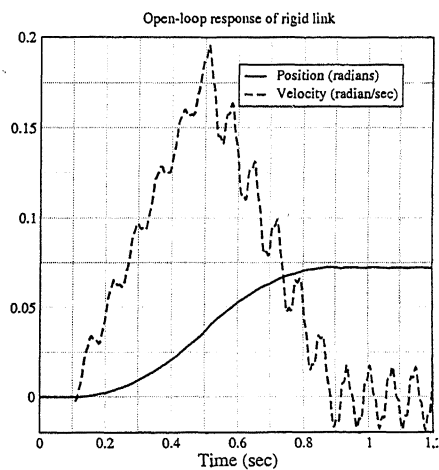
Introduce a small scale factor ε and define ξ and \tilde{K}_{ff} by

$$\varepsilon^2 \xi = q_f, \quad \tilde{K}_{ff} = \varepsilon^2 K_{ff} \quad (5.68)$$

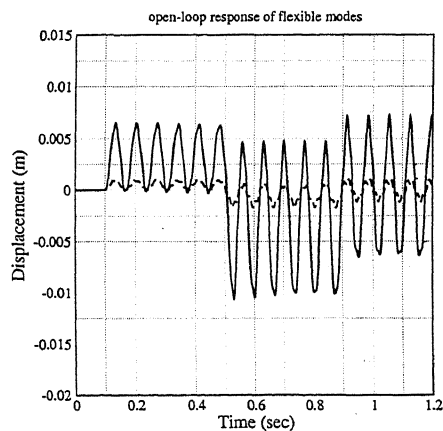
where $1/\varepsilon^2$ is equal to the smallest stiffness in K_{ff} . Substitution of (5.68) into (5.66) and (5.67) gives the system in the form,

$$\ddot{q}_r = -V_{rr}^1 \dot{q}_r - V_{rf}^1 \varepsilon^2 \dot{\xi} - (1/\varepsilon^2) H_{rf} \tilde{K}_{ff} \varepsilon^2 \xi - F_r^1 - G_r^1 + B_r^1 \tau \quad (5.69)$$

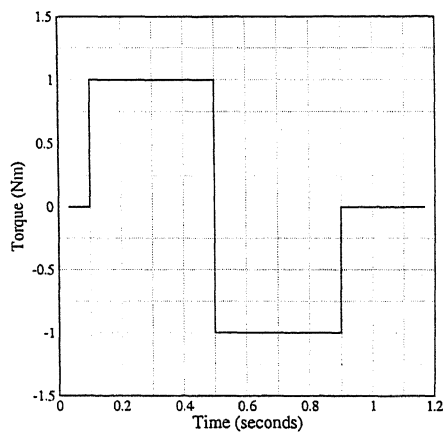
$$\varepsilon^2 \ddot{\xi} = -V_{fr}^1 \dot{q}_r - V_{ff}^1 \varepsilon^2 \dot{\xi} - (1/\varepsilon^2) H_{ff} \tilde{K}_{ff} \varepsilon^2 \xi - F_f^1 - G_f^1 + B_f^1 \tau \quad (5.70)$$



(a) Position (continuous line) and velocity (dashed line) of rigid link



(b) Flexible modes



(c) Torque profile

Figure 5.7: Open-loop response of flexible arm

where K_{ff} is invertible because it is diagonal. Define now the control

$$\tau = \bar{\tau} + \tau_F \quad (5.71)$$

with $\bar{\tau}$ the slow control component and τ_F a fast component. Setting $\varepsilon = 0$ yields

$$\ddot{q}_r = -\bar{V}_{rr}^1 \dot{q}_r - \bar{H}_{rf} \bar{K}_{ff} \bar{\xi} - \bar{F}_r^1 - \bar{G}_r^1 + \bar{B}_r^1 \bar{\tau} \quad (5.72)$$

$$\bar{\xi} = \bar{K}_{ff}^{-1} \bar{H}_{ff}^{-1} (-\bar{V}_{fr}^1 \dot{q}_r - \bar{F}_f^1 - \bar{G}_f^1 + \bar{B}_f^1 \bar{\tau}) \quad (5.73)$$

(5.72) is the slow dynamic equation and (5.73) is the slow manifold equation. Substituting (5.73) into (5.72), one can obtain the slow subsystem given by

$$\begin{aligned} \ddot{q}_r &= (\bar{H}_{rr} - \bar{H}_{rf} \bar{H}_{ff}^{-1} \bar{H}_{fr}) [-\bar{V}_{rr} \dot{q}_r - \bar{F}_r - \bar{G}_r + \bar{B}_r \bar{\tau}] \\ &= \bar{M}_{rr}^{-1} [-\bar{V}_{rr} \dot{q}_r - \bar{F}_r - \bar{G}_r + \bar{B}_r \bar{\tau}] \end{aligned} \quad (5.74)$$

To derive the fast subsystem, select states $\zeta_1 = \xi - \bar{\xi}$, $\zeta_2 = \varepsilon \dot{\xi}$ and (5.70) can be written as

$$\begin{aligned} \varepsilon \dot{\zeta}_1 &= \zeta_2 \\ \varepsilon \dot{\zeta}_2 &= -\bar{V}_{fr}^1 \dot{q}_r - \bar{V}_{ff}^1 \varepsilon \dot{\zeta}_2 - \bar{H}_{ff} \bar{K}_{ff} (\zeta_1 + \bar{\xi}) - \bar{F}_f^1 - \bar{G}_f^1 + \bar{B}_f^1 \tau \end{aligned} \quad (5.75)$$

Now making a time-scale change of $T = t/\varepsilon$ and observing that $d\xi/dT \approx 0$ and $\varepsilon \approx 0$, the fast dynamics is given by

$$\begin{aligned}\frac{d\zeta_1}{dT} &= \zeta_2 \\ \frac{d\zeta_2}{dT} &= -\bar{H}_{ff}\bar{K}_{ff}\zeta_1 + \bar{B}_f^1\tau_F\end{aligned}\quad (5.76)$$

It is important to note that this is a *linear system*, with the slow variables as parameters.

A number of control techniques for this linear system can be designed. The fast control can be chosen as

$$\tau_F = -[K_{pF} \ K_{dF}] \begin{bmatrix} \zeta_1 \\ \zeta_2 \end{bmatrix} = -\frac{K_{pf}}{\varepsilon^2}q_f - \frac{K_{dF}}{\varepsilon}\dot{q}_f + K_{pF}\bar{\xi} \quad (5.77)$$

with $\bar{\xi}$ given by (5.73). In terms of filtered tracking error, the slow subsystem (5.74) can be rewritten as

$$\bar{M}_{rr}\dot{r} = -\bar{V}_{rr}r - \bar{B}_r\bar{\tau} + f \quad (5.78)$$

where the unknown nonlinear robot function is

$$f(X) = \bar{M}_{rr}(\ddot{q}_d + \Lambda\dot{e}) + \bar{V}_{rr}(\dot{q}_d + \Lambda e) + \bar{F}_r + \bar{G}_r \quad (5.79)$$

and one may select

$$X = [e \ \dot{e} \ q_d \ \dot{q}_d \ \ddot{q}_d]^T$$

Stable NN controller: A control torque for the slow subsystem is defined as

$$\bar{\tau} = \bar{B}_r^{-1}[\hat{f} + K_v r - \nu] \quad (5.80)$$

with $\hat{f}(X)$ an estimate of $f(x)$, a gain matrix $K_v = K_v^T > 0$ and a function $\nu(t)$ that provides robustness. The closed-loop system for rigid dynamics can now be written as

$$\bar{M}_{rr}\dot{r} = -(K_v + \bar{V}_{rr})r + \tilde{f} + \nu \quad (5.81)$$

where functional estimation error is given by $\tilde{f} = f - \hat{f}$. A 2-layer NN is used to estimate the nonlinear function $f(x)$. The activation function of neurons is sigmoidal. The weights for first layer is V and for second layer it is W . The weight tuning algorithm [26] is given as

$$\begin{aligned} \dot{\hat{W}} &= F\sigma(\hat{V}^T X)r^T - F\hat{\sigma}'\hat{V}^T Xr^T - \kappa F\|r\|\hat{W} \\ \dot{\hat{V}} &= GX(\hat{\sigma}^T \hat{W}r)^T - \kappa G\|r\|\hat{V} \end{aligned} \quad (5.82)$$

The robustifying signal is given by

$$\nu(t) = -K_z(\|\hat{Z}\| + Z_B)r \quad (5.83)$$

where $Z = \text{diag}\{W, V\}$ and $\|Z\|_F \leq Z_B$. The NN control structure is shown in Figure 5.8. The response of flexible arm for $\epsilon = 0.05$ is shown in Fig. 5.9. The response of flexible arm for sinusoidal trajectory tracking is shown in Fig. 5.10.

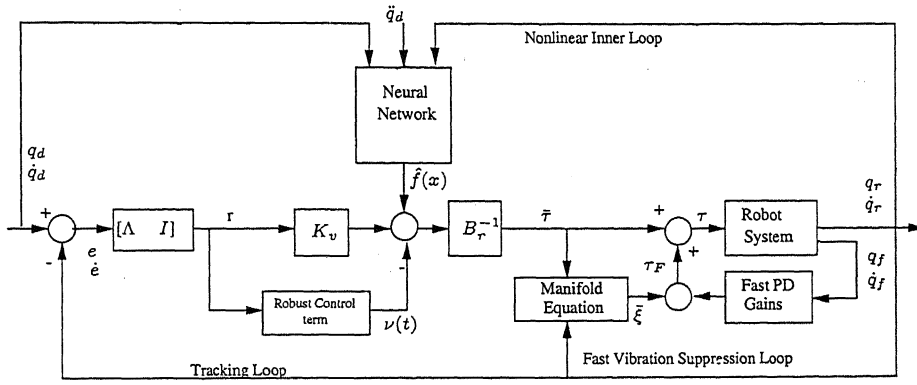


Figure 5.8: Neural Net controller for Flexible-Link robot arm

Remarks:

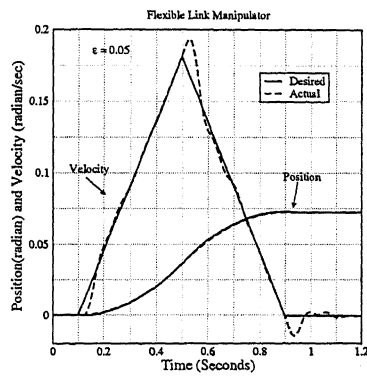
1. SPD allows splitting up a system into subsystems for which control can be designed independently from each other.
2. As can be seen from figures, Neural Network does improve the tracking by suppressing vibrations due to flexible modes.
3. Neural Networks help in approximating unmodelled dynamics and uncertain parameters inherent in the system.

5.5.4 Rigid-Link Flexible-Joint Manipulator

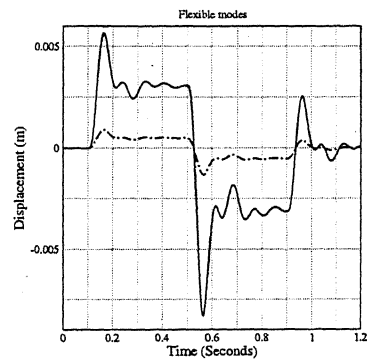
The model for an n-link RLFJ robot [25] is given by

$$M(q)\ddot{q} + V_m(q, \dot{q})\dot{q} + G(q) + F(\dot{q}) + T_L + K(q - q_f) = 0 \quad (5.84)$$

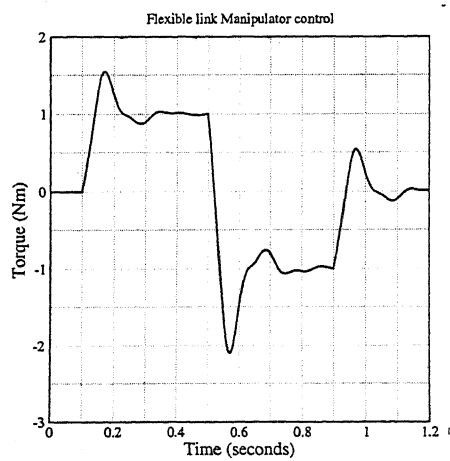
$$J\ddot{q}_m + B\dot{q}_m - K(q - q_m) = u + T_E \quad (5.85)$$



(a) position and velocity tracking by rigid link

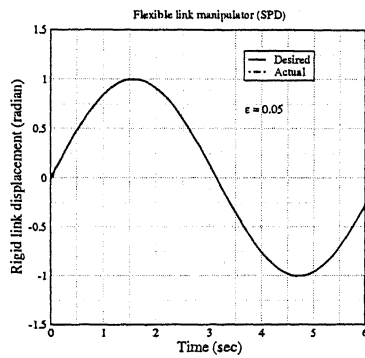


(b) flexible modes

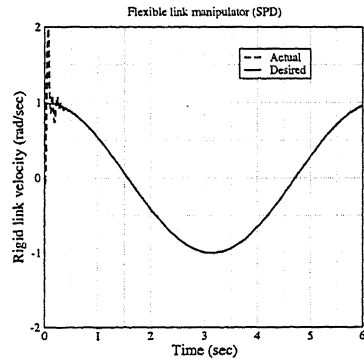


(c) Control torques

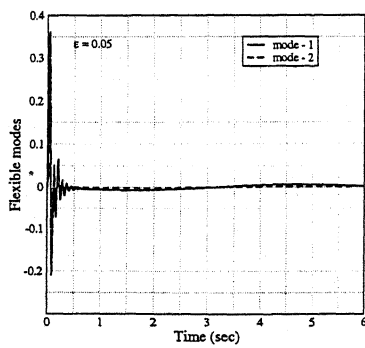
Figure 5.9: Closed loop response of flexible link manipulator with NN based SPD control



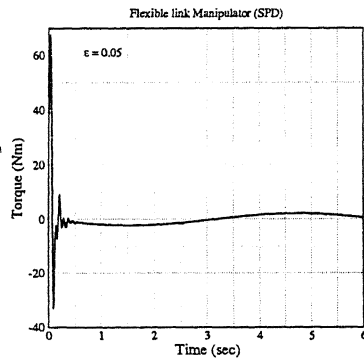
(a) Position Tracking



(b) Velocity Tracking



(c) flexible modes



(d) Control torque

Figure 5.10: Closed loop response of flexible link manipulator for sinusoidal trajectory

with $q, \dot{q}, \ddot{q} \in R^n$ denoting the link position, velocity and acceleration vectors, respectively, $M(q) \in R^{n \times n}$, the inertia matrix, $V_m(q, \dot{q}) \in R^n$, the centripetal-coriolis matrix, $G(q) \in R^n$, the gravity vector, $F(\dot{q}) \in R^n$ representing friction terms, $T_L \in R^n$ the additive disturbance, $q_m, \dot{q}_m, \ddot{q}_m \in R^n$, the motor shaft angle, velocity, acceleration, respectively. $K \in R^{n \times n}$ the positive definite constant diagonal matrix which characterizes joint flexibility, $J \in R^{n \times n}$ a positive definite constant diagonal matrix denoting motor inertia, $B \in R^{n \times n}$ represents natural damping term, u the control vector used to represent the motor torque, and $T_E \in R^n$ representing an additive bounded torque disturbance. The rigid dynamics (5.84) has two important properties (5.1) and (5.2), stated in section 5.2. The joint elasticity matrix K is bounded by

$$k_1 \|x\|^2 \leq x^T K x \leq k_2 \|x\|^2, \text{ for arbitrary vector } x$$

where k_1 and k_2 are positive scalar bounding constants. The motor inertia matrix J is also bounded by

$$l_1 \|x\|^2 \leq x^T J x \leq l_2 \|x\|^2, \text{ for arbitrary vector } x$$

where l_1 and l_2 are positive scalar bounding constants.

Control Objective: The control objective for the flexible-joint robot arm is to control the arm joint variable $q(t)$ in the face of the additional dynamics of the variable $q_m(t)$, the motor angle. This is similar to the situation for the flexible-link arm (5.62), where one must control the arm in the face of the additional dynamics of $q_f(t)$, the flexible

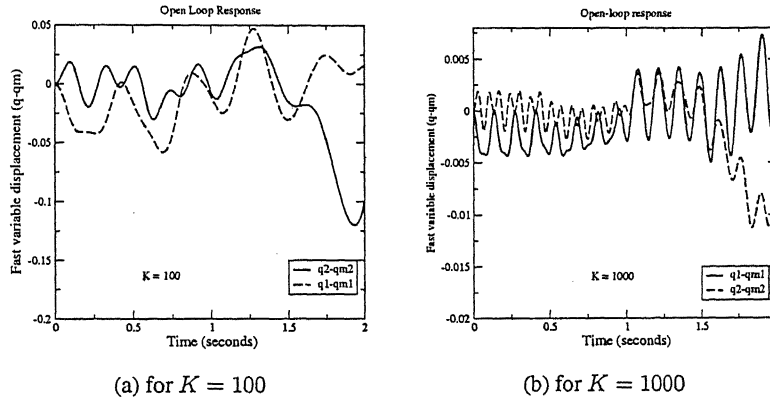


Figure 5.11: Open-loop response of fast variable $q - q_m$

modes. However, control design for these two problems must be approached by different philosophies. The flexible-link arm is fundamentally a disturbance rejection problem, while for flexible-joint arm one is faced with manipulating an intermediate variable $q_m(t)$ that has subsequent influence on the variable of interest $q(t)$.

Singular Perturbation Design for 2-Link RLFJ manipulator

Before one starts with singular perturbation design, one must see whether it is possible to split the system into two independent subsystems namely fast and slow subsystems. Since the dynamics of robot arm and the motor are in almost same time scale, we choose $q - q_m$ as our fast variable. The open-loop response of $q - q_m$ for $K = 100$ and $K = 1000$ is shown in the Fig. 5.11. As one can see, for higher value of K , the frequency of fast variable increases and thus one has a better chance of splitting the given system into fast/slow subsystems. Lets choose $\epsilon^2 \xi = q - q_m$ and $\tilde{K} = \epsilon^2 K$. We can write (5.84)-(5.85) as

follows:

$$0 = M\ddot{q} + V_m\dot{q} + G + F + \tilde{K} \quad (5.86)$$

$$J\varepsilon^2\ddot{\xi} = J\ddot{q} + B\dot{q} - B\varepsilon^2\dot{\xi} - \tilde{K} - u \quad (5.87)$$

The slow dynamics can be obtained by substituting $\varepsilon = 0$ in above equations. The slow manifold equation is obtained from (5.87) as

$$\bar{\xi} = \tilde{K}^{-1}(J\ddot{q} + B\dot{q} - \bar{u}) \quad (5.88)$$

where $\bar{\xi}$, \bar{u} and \bar{q} represent slow variables. Substituting $\bar{\xi}$ from (5.88) and $\varepsilon = 0$ into (5.86), one can write the slow dynamics as

$$\begin{aligned} (M + J)\ddot{\bar{q}} + (V_m + B)\dot{\bar{q}} + G + F &= \bar{u} \\ \bar{M}\ddot{\bar{q}} + \bar{V}\dot{\bar{q}} + G + F &= \bar{u} \end{aligned} \quad (5.89)$$

where $\bar{M} = M + J$ and $\bar{V} = V_m + B$. The slow subsystem is of reduced order. By defining two state variables $\zeta_1 = \xi - \bar{\xi}$ and $\zeta_2 = \varepsilon\dot{\xi}$, the state equations for fast subsystem may be written as

$$\begin{aligned} \varepsilon\dot{\zeta}_1 &= \zeta_2 \\ \varepsilon\dot{\zeta}_2 &= \ddot{\bar{q}} + J^{-1}B\dot{\bar{q}} - J^{-1}B\varepsilon^2\dot{\xi} - J^{-1}\tilde{K}\xi - J^{-1}u \end{aligned} \quad (5.90)$$

Substituting $u = \bar{u} + \tilde{u}$, $\xi = \zeta_1 + \bar{\xi}$ and $\bar{\xi}$ from (5.88) into (5.90), we get

$$\epsilon \dot{\zeta}_2 = -J^{-1}B\epsilon\zeta_2 - J^{-1}\tilde{K}\zeta_1 - J^{-1}\tilde{u} \quad (5.91)$$

Now, changing the time scale $T = t/\epsilon$ and assuming that $\epsilon \approx 0$, we get the fast subsystem

$$\begin{aligned} \frac{d\zeta_1}{dT} &= \zeta_2 \\ \frac{d\zeta_2}{dT} &= -J^{-1}\tilde{K}\zeta_1 - J^{-1}\tilde{u} \end{aligned} \quad (5.92)$$

This is a linear system which can be stabilized using any of the linear system techniques like LQR or state-feedback. The control input for the fast system is given as

$$\tilde{u} = \frac{K_{pf}}{\epsilon^2}(q - q_m) + \frac{K_{df}}{\epsilon}(\dot{q} - \dot{q}_m) - K_{pf}\bar{\xi} \quad (5.93)$$

The slow subsystem is a nonlinear system which is now controlled as follows:

Define $e = q_d - q$, $r = \dot{e} + \Lambda e$. The slow subsystem (5.89) can be written in terms of filtered error as,

$$\begin{aligned} \bar{M}\dot{r} &= \bar{M}(\ddot{q}_d + \Lambda\dot{e}) + \bar{V}\dot{q} + G + F - \bar{u} \\ &= \bar{M}(\ddot{q}_d + \Lambda\dot{e}) + \bar{V}(\dot{q}_d + \Lambda e) + G + F - \bar{V}r - \bar{u}(\ddot{q}_d + \Lambda\dot{e}) + \bar{V}(\dot{q}_d + \Lambda e) + G + F \\ &= F_1 - \bar{V}r - \bar{u} \end{aligned} \quad (5.94)$$

where

$$F_1 = \bar{M}(\ddot{q}_d + \Lambda \dot{e}) + \bar{V}(\dot{q}_d + \Lambda e) + G + F$$

Lets choose a control

$$\bar{u} = K_v r + \hat{F}_1 - \nu \quad (5.95)$$

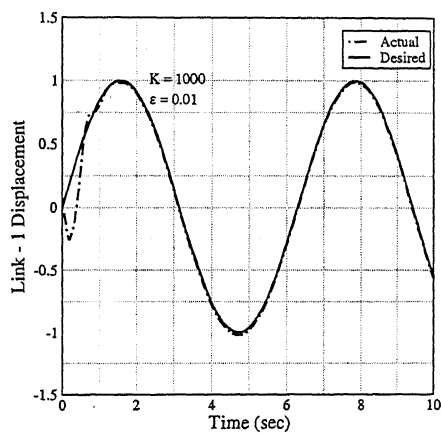
where \hat{F}_1 is the approximation of F_1 obtained from a 2-layer NN. ν is the robustifying term (5.83). The closed-loop response of the slow system is given as follows:

$$\bar{M}\dot{r} = (F_1 - \hat{F}_1) - (\bar{V} + K_v)r + \nu \quad (5.96)$$

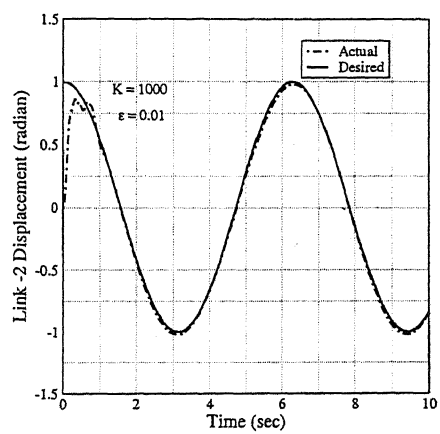
The weight update algorithm (5.82) ensures the stability of above closed loop system. Thus we obtain two independent control inputs \bar{u} from (5.95) and \tilde{u} from (5.93) and the final control input is given as $u = \tilde{u} + \bar{u}$. The simulation results for $K = 1000$ are shown in Fig. 5.12. Here $K_{pf} = \text{diag}\{0.37, 0.37\}$ and $K_{dF} = \text{diag}\{0.69, 0.69\}$.

Remarks:

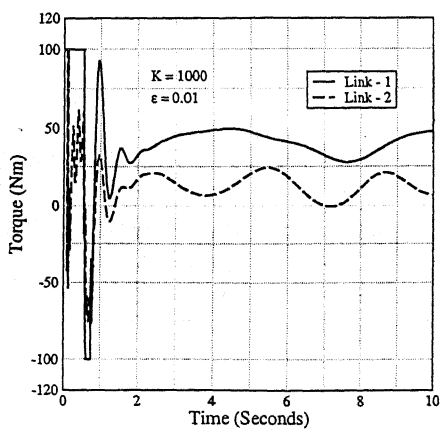
- Lower value of stiffness coefficient matrix K necessitates high control input. So, the stiffness matrix should be very high, the reason being the fact that for highly flexible system, we can't split the system into fast/slow subsystem.
- The gain matrices in (5.93) K_{pf} and K_{dF} are very small. This method does not require high feedback gains which is certainly desirable.
- NN approximation obviates the necessity for exact computation of nonlinear and uncertain terms thereby simplifying the control problem.



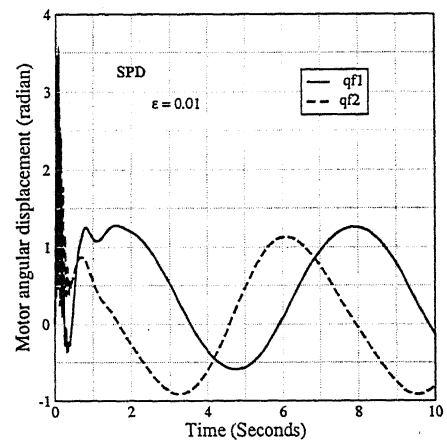
(a) Link 1 Position Tracking



(b) Link 2 Position Tracking



(c) Control Torque



(d) Motor rotor position

Figure 5.12: Simulation Results of NN based SPD control of 2-Link Flexible-Joint Manipulator

5.6 Summary

In the previous chapters, we studied how Neural Networks can be used for identifying nonlinear systems. In this chapter, we study various neural network based controllers for nonlinear as well as underactuated systems. For fully actuated systems, we have many control techniques namely, feedback linearization, backstepping and other robust and adaptive control techniques etc. The control strategies become more complex when the number of actuators become less than the degrees of freedom. The problem with these techniques is that they require certain simplifying assumptions which limit their applicability and in many practical situations, these assumptions don't hold good. Moreover, the computation of regression matrices in adaptive controllers is very formidable. We saw that the use of Neural Network circumvents these problems and don't require any such restricting assumptions. Thus, NN based controllers can be applied to a wider class of problems. Two NN based controllers namely, NN-Backstepping and NN-singular perturbation have been studied in detail and their application to various underactuated systems like, flexible-link and flexible joint robot manipulators have been demonstrated.

Chapter 6

The Pendubot

6.1 Introduction

The pendubot is a two-link planar robot with an actuator at the shoulder (link 1) and no actuator at the elbow (link 2). The link 2 moves freely around link 1 and the control objective is to bring the mechanism to the unstable equilibrium point. Block [29] proposed a control strategy based on two control algorithms to control the pendubot. For swing up control, he used partial feedback linearization technique and for balancing control, he linearized the system about the top equilibrium point and used Linear Quadratic Regulator (LQR) or pole placement techniques to stabilize at the top position. Fantoni and Lozano [27] utilized the passivity properties of pendubot and used an energy-based approach to propose a control law. They also provided complete stability analysis of their method. In this chapter, I tried to analyze Block's method and made use of Neural Networks to stabilize the pendubot at its top unstable equilibrium point. Since this is an underactu-

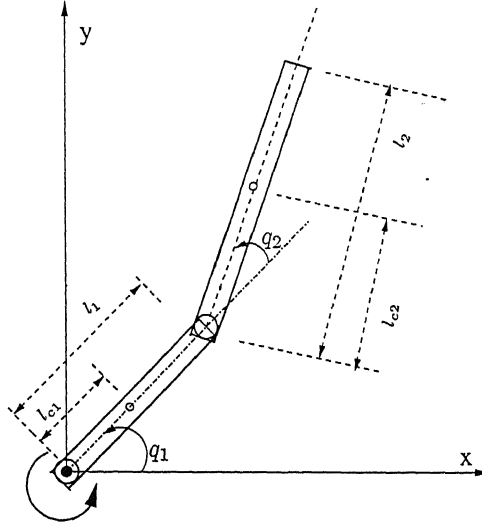


Figure 6.1: The pendubot system

ated system, one need to consider the energy aspects of the pendubot while designing a controller.

6.2 NN based Partial Feedback Linearization

Consider the pendubot as shown in the figure 6.3. The equations of motion for the Pendubot can be found using Lagrangian dynamics [27]. The equations can be written in matrix form as follows:

$$D(q)\ddot{q} + C(q, \dot{q})\dot{q} + g(q) = \tau \quad (6.1)$$

where

$$q = \begin{bmatrix} q_1 \\ q_2 \end{bmatrix} \quad D(q) = \begin{bmatrix} \theta_1 + \theta_2 + 2\theta_3 \cos(q_2) & \theta_2 + \theta_3 \cos(q_2) \\ \theta_2 + \theta_3 \cos(q_2) & \theta_2 \end{bmatrix}$$

$$C(q, \dot{q}) = \begin{bmatrix} -\theta_3 \sin(q_2) \dot{q}_2 & -\theta_3 \sin(q_2) \dot{q}_1 \\ \theta_3 \sin(q_2) \dot{q}_1 & 0 \end{bmatrix}$$

$$g(q) = \begin{bmatrix} \theta_4 g \cos(q_1) + \theta_5 g \cos(q_1 + q_2) \\ \theta_5 g \cos(q_1 + q_2) \end{bmatrix} \quad \tau = \begin{bmatrix} \tau_1 \\ 0 \end{bmatrix}$$

and

$$\theta_1 = m_1 l_{c1}^2 + m_2 l_1^2 + I_1 \quad \theta_2 = m_2 l_{c2}^2 + I_2$$

$$\theta_3 = m_2 l_1 l_{c2} \quad \theta_4 = m_1 l_{c1} + m_2 l_1 \quad \theta_5 = m_2 l_{c2}$$

The various variables used above are: m_1 and m_2 are the masses of link 1 and 2 respectively, l_1 and l_2 are their corresponding lengths, l_{c1} and l_{c2} are the distances to the center of masses of the respective links, I_1 and I_2 are the moments of inertia of the two links about their centroids.

6.3 Swing Up Control

The equations of motion of Pendubot are given by (6.1). Performing the matrix and vector multiplication, the equations of motion are written as

$$\tau_1 = d_{11} \ddot{q}_1 + d_{12} \ddot{q}_2 + c_{11} \dot{q}_1 + c_{12} \dot{q}_2 + g_1 \quad (6.2)$$

$$0 = d_{21} \ddot{q}_1 + d_{22} \ddot{q}_2 + c_{21} \dot{q}_1 + g_2 \quad (6.3)$$

Due to the underactuation of link 2, we can not linearize dynamics about both degrees of freedom. We can however linearize one of the degrees of freedom. In the case of Pendubot, we linearize about q_1 . The equation (6.3) is solved for link 2's acceleration

$$\ddot{q}_2 = \frac{-d_{21}\dot{q}_1 - c_{21}\dot{q}_1 - g_2}{d_{22}}$$

Substituting \ddot{q} from the above equation into (6.2), we get

$$\tau_1 = \bar{d}_{11}\ddot{q}_1 + \bar{c}_{11}\dot{q}_1 + \bar{c}_{12}\dot{q}_2 + \bar{g}_1 \quad (6.4)$$

with

$$\begin{aligned} \bar{d}_{11} &= d_{11} - \frac{d_{12}d_{21}}{d_{22}} & \bar{c}_{11} &= \frac{d_{12}c_{21}}{d_{22}} \\ \bar{c}_{12} &= c_{12} & \bar{g}_1 &= g_1 - \frac{d_{12}g_2}{d_{22}} \end{aligned}$$

In terms of filtered error, (6.4) can be written as

$$\begin{aligned} \bar{d}_{11}\dot{r} &= \bar{d}_{11}(\ddot{q}_{d1} + \lambda\dot{e}) + \bar{c}_{11}(\dot{q}_{d1} + \lambda e) + \bar{c}_{12}\dot{q}_2 + g_1 - \bar{c}_{11}r - \tau_1 \\ &= F - \bar{c}_{11}r - \tau_1 \end{aligned} \quad (6.5)$$

where

$$r = \dot{e} + \lambda e, \quad e = q_{d1} - q_1$$

$$F = \bar{d}_{11}(\ddot{q}_{d1} + \lambda\dot{e}) + \bar{c}_{11}(\dot{q}_{d1} + \lambda e) + \bar{c}_{12}\dot{q}_2 + g_1$$

The nonlinear function F can also include friction terms which we have neglected in the model (6.1). Moreover, various pendubot parameters may not be known quite accurately.

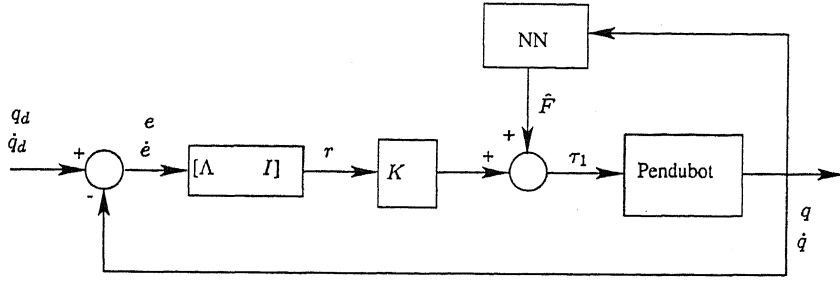


Figure 6.2: NN Controller for Pendubot

We can approximate this nonlinear and uncertain function using a 2-layer Neural Network [25, 39]. The NN controller is shown in Fig. 6.2. A control input can be chosen as

$$\tau_1 = \hat{F} + k_1 r \quad (6.6)$$

The closed loop dynamics for q_1 is given as

$$\bar{d}_{11} \dot{r} = (F - \hat{F}) - (\bar{c}_{11} + k_1) r \quad (6.7)$$

The closed loop dynamics is stable provided $\|F - \hat{F}\| \leq \varepsilon$. The weight update algorithm proposed by Kwan et al [25], ensures the boundedness of weights as well as error signal $(F - \hat{F})$. If the NN approximation holds good then one can see that the closed-loop dynamics (6.7) becomes a linear one. Since only one link is being linearized, it can be considered as *partial feedback linearization*.

By giving a step trajectory, it is easy to bring the first link to the top position, but we need to pump sufficient energy to the second link so that it rotates around its own axis.

With this objective in mind, we choose following swing up trajectory for the link 1

$$q_{d1} = \begin{cases} 4\pi & \text{for } t \leq 2\pi \\ \pi/2 & \text{for } t > 2\pi \end{cases}$$

Once the link 2 comes closer to the top unstable equilibrium point we switch over to some linear control as mentioned next.

6.4 Balancing Control

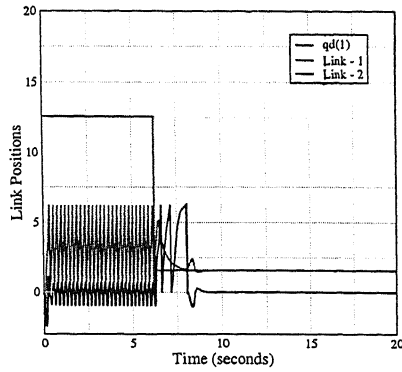
We linearize the Pendubot's nonlinear equations of motion about the top unstable equilibrium position ($q_1 = \pi/2, q_2 = 0$). The linear model for the top position is as follows

$$\dot{x} = Ax + Bu \tag{6.8}$$

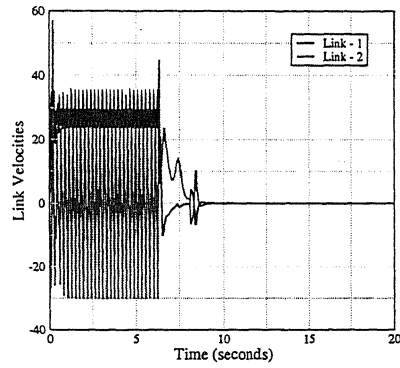
where

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 51.9265 & 0 & -13.9704 & 0 \\ 0 & 0 & 0 & 1 \\ -52.8402 & 0 & 68.4210 & 0 \end{bmatrix} \quad B = \begin{bmatrix} 0 \\ 15.9549 \\ 0 \\ -29.3596 \end{bmatrix}$$

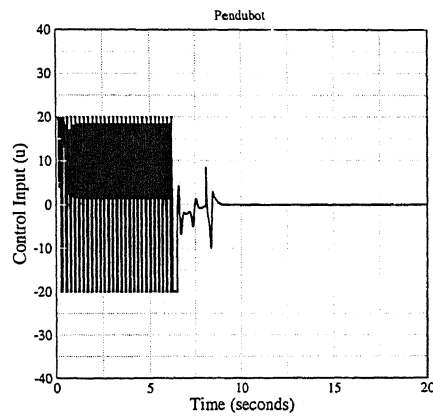
By using pole-placement technique, we design a state feedback controller $u = -Kx$ to stabilize the system around its equilibrium point.



(a) Position Tracking



(b) Velocity Tracking



(c) Control Torque

Figure 6.3: NN based Partial Feedback Linearization control of Pendubot

6.5 Simulation

The pendubot parameters are $\theta_1 = 0.0799$, $\theta_2 = 0.0244$, $\theta_3 = 0.0205$, $\theta_4 = 0.42126$, $\theta_5 = 0.10630$ and $g = 9.8$. The swing up control (6.6) is used to bring the link 1 to the top position, once the link 2 reaches close to the top position ($|q_{d1} - q_1| \leq 0.2$, $|q_{d2} - q_2| \leq 0.2$), we switch over to a linear state feedback controller where the state feedback gain is given by

$$K = [-32.68 \quad -7.14 \quad -32.76 \quad -4.88]$$

The experimental results are shown in figure 6.3.

Remarks:

- Neural Networks help in taking unmodelled dynamics like friction into account while designing controller and thus simplifies the numerical computation required.
- Control input is bounded.
- The initial trajectory for link 1 has been decided heuristically. One has to make several trials with the real system to come up with a suitable trajectory which will take the pendubot to its upright position in a smooth manner. Fantoni et al [27], have proposed a energy based method, where they bring the second link into *homoclinic orbit*. One may investigate along that line to come up with a smooth trajectory.

6.6 Summary

The pendubot is an example of underactuated system where the number of degrees of freedom is less than the number of actuators. A controller based on NN is proposed in this chapter. The NN is used to approximate the inherent nonlinearities and unmodelled dynamics of the system thereby relieving the designer from tedious mathematical computations. Kwan's weight update algorithm [25] ensures the boundedness of weights as well as tracking error. This controller has some resemblance to Block's work [29], in the sense that we too use partial feedback linearization for link 1. In his work, two loops have been used for swing up control, the outer-loop being a PD control with feedforward acceleration which achieves the trajectory tracking of link 1 and the inner-loop is partial feedback linearization loop which linearizes the link 1 dynamics. He too selected a trajectory in the beginning for link 1 in order to excite the link 2 dynamics which was based on his experience with his hardware set up. In our case, we have only one control loop. The shortcoming of our algorithm is that one has to try a number of trajectories in the initial stages which not only excites the link 2 but also brings it closer to the top position. Further investigations may be done in this aspect to make it a robust control strategy.

Chapter 7

Conclusion

7.1 Contributions

In this thesis, system identification and control of nonlinear systems using neural networks have been studied and analyzed. Some of the contributions of this work are as follows:

- Extended Kalman Filtering (EKF) has been used to train a Memory Neuron Network (MNN) for the first time. Performance comparison among three algorithms namely back propagation through time (BPTT), real time recurrent learning (RTRL) and EKF for this network has been done in identifying both SISO as well as MIMO systems.
- A novel learning algorithm based on Lyapunov stability theory has been proposed for feedforward networks. Its performance has been compared with existing BP and EKF algorithms. The nature of adaptive learning rate has been analyzed in de-

tail. A modification has also been suggested to speed up the convergence. Through simulations, the proposed algorithm has been shown to give faster convergence. Various system identification issues have also been discussed for the proposed algorithm.

- Some of the recent NN based adaptive and robust control techniques have been studied in detail. Two methods namely NN based robust backstepping control and NN control based on singular perturbation have been used to control various robot manipulators. Through simulation results, it is shown that NN controllers give substantial improvement in the performance.
- A new NN control based on partial feedback linearization has been suggested for pendubot. It is shown that some of the unmodelled dynamics like friction can be taken into consideration by using NN.

7.2 Scope of Future work

- Finding a simple learning algorithm for updating weights in neural networks, which is faster in terms of convergence time and computational complexity, is still an open research problem. A lot of modifications can be brought about both in the architecture as well as the learning algorithm to achieve faster convergence.
- Usually, feedforward networks are used for controlling nonlinear dynamical systems like robot manipulators, owing to its ease of implementation. But feedforward networks have certain limitations, for instance, they need complete knowl-

edge about the states of the system. It is seen that MNN structure provide the capability of RNN along with the simplicity of MLP. Thus MNN may be used for implementing online controllers for robot manipulators.

- The proposed LF algorithm has been shown to be fast and easy to implement. Here, this algorithm has been applied only for system identification. So, one may also use it to design online controllers.
- The suggested NN based control for pendubot need to be tested on a hardware set up.
- Neural Network and machine learning approaches have not been applied to under-actuated mechanical systems. It will be interesting to see if these concepts can be used to design meaningful controllers for this class of nonlinear systems.

Appendix A

Definitions and theorems

A.1 Barbalat's lemma

If the differentiable function $f(t)$ has a finite limit as $t \rightarrow \infty$, and if \dot{f} is uniformly continuous, then $\dot{f}(t) \rightarrow 0$ as $t \rightarrow \infty$.

A function is said to be *uniformly continuous* on $[0, \infty)$ if $\forall R > 0, \exists \eta(R) > 0, \forall t_1 > 0, \forall t > 0, |t - t_1| < \eta \Rightarrow |g(t) - g(t_1)| < R$. A sufficient condition for a differentiable function to be uniformly continuous is that its *derivative* be bounded.

Corollary A.1 *if the differentiable function $f(t)$ has a finite limit as $t \rightarrow \infty$, and is such that \ddot{f} exists and is bounded, then $\dot{f}(t) \rightarrow 0$ as $t \rightarrow \infty$.*

A.2 Strictly Positive Real Systems

A transfer function $h(p)$ is *positive real* if

$$\operatorname{Re}[h(p)] \geq 0 \quad \forall \operatorname{Re}[p] > 0$$

It is *strictly positive real* if $h(p - \epsilon)$ is positive real for some $\epsilon > 0$.

Theorem A.1 A transfer function $h(p)$ is strictly positive real (SPR) if and only if

- $h(p)$ is a strictly stable transfer function
- the real part of $h(p)$ is strictly positive along the $j\omega$ axis, i.e., $\forall \omega \geq 0, \operatorname{Re}[h(j\omega)] > 0$

The above theorem implies following *necessary* conditions for asserting whether a given transfer function $h(p)$ is SPR:

- $h(p)$ is strictly stable
- The Nyquist plot of $h(j\omega)$ lies entirely in the right half complex plane.
- $h(p)$ has a relative degree of 0 or 1
- $h(p)$ is strictly minimum phase

A.3 Zero Dynamics

The part of system dynamics, which can not be seen from external input-output relationship, is known as the *internal dynamics* of a system. For linear systems, the stability of

the internal dynamics is simply determined by the location of zeros. This concept can not be extended to nonlinear systems.

The *zero dynamics* is defined to be the internal dynamics of the system when the system output is kept at zero by the input. Two useful remarks can be made about the zero dynamics of nonlinear systems.

- the zero dynamics is an intrinsic feature of a nonlinear system, which does not depend on the choice of control law or the desired trajectories.
- the stability of internal dynamics of nonlinear system can be judged by examining the stability of zero dynamics.

A system with unstable zero dynamics is called a *non-minimum-phase* system.

A.4 Persistent Excitation

By persistent excitation of a signal vector $v(t)$, we mean that there exist strictly positive constants α_1 and T such that for any $t > 0$,

$$\int_t^{t+T} v^T v dr \geq \alpha_1 I$$

Intuitively, the persistent excitation of $v(t)$ implies that the vectors $v(t)$ corresponding to different times t cannot always be linearly dependent.

Bibliography

- [1] R. P. Lippmann. An introduction to computing with neural networks. *IEEE ASSP Magazine*, 4(2), April 1987.
- [2] K. S. Narendra and K. Parthasarathy. Identification and control of dynamical systems using neural networks. *IEEE Transaction on Neural Networks*, 1(1):4–27, 1990.
- [3] F. Chen and H. K. Khalil. Adaptive control of nonlinear systems using neural networks. *International Journal Of Control*, 55(6):1299–1317, 1992.
- [4] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. *Learning internal representations by error propagation*. MIT Press, Cambridge, MA, USA, 1986.
- [5] P. J. Werbos. Back propagation through time: What it does and how to do it. In *IEEE Proc.*, volume 78, pages 1550–1560, October 1990.
- [6] R. J. Williams and D. Zipser. Gradient-based learning algorithms for recurrent connectionist networks. Technical report, College of Computer Science, Northeastern University, Boston, 1990.
- [7] Stanislaw Osowski, Piotr Bojarczak, and Maciej Stodolski. Fast second order learning algorithm for feedforward multilayer neural network and its applications. *Neural Networks*, 9(9):1583–1596, 1996.
- [8] Jaroslaw Bilski and Leszek Rutkowski. A fast training algorithm for neural networks. *IEEE Transactions on Circuits and Systems-II: Analog and Digital signal processing*, 45(6):749–753, june 1998.
- [9] Bogdan M. Wilamowski, Serder Iplikci, Okay Kaynak, and M. Onder Efe. An algorithm for fast convergence in training neural networks. *IEEE*, 2001.
- [10] Martin T. Hagan and Mohammad B. Mehnaj. Training feedforward networks with marquardt algorithm. *IEEE Transaction on Neural Networks*, 5(6):989–993, November 1994.
- [11] G. Lera and M. Pinzolas. Neighborhood based levenberg-marquardt algorithm for neural network training. *IEEE Transactions on Neural Networks*, 13(5):1200–1203, september 2002.

- [12] C. Charalambous. Conjugate gradient algorithm for efficient training of artificial neural networks. In *IEE Proceedings*, volume 139, pages 301–310, 1992.
- [13] Youji Iiguni, Hideaki Sakai, and Hidekatsu Tokumaru. A real time learning algorithm for a multilayered neural network based on extended kalman filter. *IEEE Transaction on Signal Processing*, 40(4):959–966, 1992.
- [14] Gintaras V. Puskorius and Lee A. Feldkamp. *Kalman Filtering and Neural Networks*. John Wiley & Sons, Inc., 2001.
- [15] M. W. Spong and M. Vidyasagar. *Robot Dynamics and Control*. Wiley, New York, 1989.
- [16] K. S. Narendra and K. Parthasarathy. Gradient methods for optimisation of dynamical systems containing neural networks. *IEEE Transaction on Neural Networks*, 2(2):252–262, 1991.
- [17] Asriel U. Levin and Kumpati S. Narendra. Control of nonlinear dynamical systems using neural networks: Controllability and stabilization. *IEEE Transactions on Neural Networks*, 4(2), March 1993.
- [18] A. Delgado, C. Kambhampati, and K. Warwick. Dynamic recurrent neural network for system identification and control. In *Control Theory Applications*, volume 142. IEE Proceedings, July 1995.
- [19] Laxmidhar Behera. Query based model learning and stable tracking of a robot arm using radial basis function network. *Computers and Electrical Engineering, Elsevier Science Ltd.*, 29:553–573, 2003.
- [20] J. J. E. Slotine and Weiping Li. Adaptive manipulator control: A case study. *IEEE Transactions on Automatic Control*, 33(11):995–1003, November 1988.
- [21] Petar V. Kokotovic. The joy of feedback: Nonlinear and adaptive. *IEEE Control System Magazine*, (3):7–17, June 1992.
- [22] M. W. Spong. On robust control of robot manipulators. *IEEE Transactions on automatic control*, 37(11):1782–1786, November 1992.
- [23] Miroslav Krstic, Ioannis Kanellakopoulos, and Petar Kokotovic. *Non Linear and Adaptive control design*. John Wiley & Sons, Inc., 1995.
- [24] J. J. E. Slotine and W. Li. *Applied Non-Linear Control*. Prentice Hall, New Jersey, 1991.
- [25] Chiman Kwan and F. L. Lewis. Robust backstepping control of nonlinear systems using neural networks. *IEEE Transactions on Systems, Man and Cybernetics - Part A*, 30(6):753–766, November 2000.

- [26] F. L. Lewis, S. Jagannathan, and A. Yesildirek. *Neural Network control of Robot Manipulators and Nonlinear Systems*. Taylor & Francis, 1999.
- [27] Isabelle Fantoni and Rogelio Lozano. *Non-linear Control for Underactuated Mechanical Systems*. Springer-Verlag, 2002.
- [28] M. W. Spong. Swing up control of the acrobot. pages 2356–2361, San Diego, CA, May 1994. IEEE International conference on Robotics and Automation.
- [29] D. J. Block. Mechanical design and control of pendubot. Master's thesis, University of Illinois, 1991.
- [30] Reza Olfati-Saber. *Nonlinear Control of Underactuated Mechanical Systems with Application to Robotics and Aerospace vehicles*. PhD thesis, Massachusetts Institute of Technology, February 2001.
- [31] P. S. Sastry, G. Santharam, and K. P. Unnikrishnan. Memory neuron networks for identification and control of dynamical systems. *IEEE Transaction on Neural Networks*, 5(2):306–319, 1994.
- [32] Laxmidhar Behera, Madan Gopal, and Santanu Choudhury. On adaptive trajectory tracking of a robot manipulator using inversion of its neural emulator. *IEEE Transaction on Neural Networks*, 7(6):1401–1414, November 1996.
- [33] M. Vidyasagar. *Nonlinear Systems Analysis*. Prentice Hall, New Jersey, 1993.
- [34] R. J. Williams. Some observations on the use of extended kalman filter as a recurrent network learning algorithm. Technical report, College of Computer Science, Northeastern University, Boston, 1992.
- [35] R. J. Williams. Training recurrent networks using the extended kalman filter. In *International joint Conference on Neural Networks*, volume II, pages 241–246, Baltimore, 1992.
- [36] Xinghuo Yu, M. Onder Efe, and Okyay Kaynak. A backpropagation learning framework for feedforward neural networks. *IEEE Transactions on Neural Networks*, 2001.
- [37] Chien-Kuo Li. A sigma-pi-sigma neural network(spsnn). *Neural Processing Letters*, 17:1–19, 2003.
- [38] Simon Haykin. *Neural Networks, A comprehensive foundation*. Prentice Hall International, Inc., 1999.
- [39] Chiman Kwan, Frank L. Lewis, and Darren M. Dawson. Robust neural-network control of rigid-link electrically driven robots. *IEEE Transactions on Neural Networks*, 9(40):581–588, July 1998.
- [40] K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(359-366), 1989.

Index

- Adaptive control, 38
 - MRAC, 39
 - NNC, 48
 - STC, 41
- ANN, 1
- Backstepping control, 59
- Barbalat's Lemma, 22, 112
- Internal dynamics, 113
- Least square estimator, 44
- LF algorithm, 20
 - 3-bit Parity, 30
 - 4-2 Encoder, 32
 - Gabor function, 34
 - LF-II, 24
 - System identification, 32
 - XOR, 28
- Linear Parametrization, 43
- LIP, 61
- MNN, 2, 6
 - Architecture, 11
 - BPTT, 6
 - Dynamics, 3
 - EKF, 9
 - RTRL, 7
- Non-minimum phase system, 114
- Nonlinear control, 3
- Pendubot, 100
 - Balancing control, 105
 - partial feedback linearization, 101
 - Swing up control, 102
- Persistent excitation, 46, 114
- Positive Real function, 113
- Regression matrices, 61
- RNN, 6
- Robot Manipulator
 - Flexible link, 81
- Robot Manipulators, 54
 - Dynamics, 55
 - Properties, 56
 - RLED, 73
 - RLFJ, 90
- Single Link Manipulator, 39
- Singular Perturbation Design, 69
- SPR, 113
- Strict feedback form, 60
- System identification, 2
- UAMS, 4
- Uniformly Continuous function, 112
- UUB, 60
- Zero dynamics, 113